DOCUMENT RESUME

ED 130 617 IR 004 148

AUTHOR Jacobs, Panl I.
TITLE Some Implications of Testing Procedures for Anto-Instructional Programming. Final Report.
INSTITUTION Educational Testing Service, Princeton, N.J.
SPONS AGENCY Air Force Human Resonrces Lab., Wright-Patterson AFB, Ohio.
REPORT NO MRL-TDR-62-67; P-1710
PUB DATE Jnn 62
CONTRACT AF-33(616)-7795
NOTE 79p.: Archival docnment

EDRS PRICE MF-$0.83 HC-$4.67 Plns Postage.
DESCRIPTORS *Antoinstrnctional Programs; Formative Evaluation; *Instrnctional Design; *Material Development; Programed Instrnction; Programing; *Test Constrnction

ABSTRACT

Althongh there are fnndamental differences in the objectives of the two activities, the programing of instrnctional materials bears many similarities to the construction of tests. A systematic comparison of the problems and procednres reveals important implications for programing from the older field of testing. Theory and experience in test construction can be especially usefnl in the selection of valid criteria for assessing the effectiveness of a program, the ordering of instrnctional snbject matter, the writing of instructional frames, and the formal evaluation of a program. Adaptive programing implies measurement of both aptitude and achievement in order to assign trainees to appropriate individnal seqnences of instrnction. Possible applications resnlting from examination of these and other issnes are explored, and necessary further research is snggested. (Anthor)

MRL-TDR-62-67

SOME IMPLICATIONS OF TESTING PROCEDURES FOR AUTO-INSTRUCTIONAL PROGRAMMING

TECHNICAL DOCUMENTARY REPORT No. MRL-TDR-62-67

JUNE 1962

BEHAVIORAL SCIENCES LABORATORY
6570th AEROSPACE MEDICAL RESEARCH LABORATORIES
AEROSPACE MEDICAL DIVISION
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO

Contract Monitor: Alvin Ugelow, Ph.D.
Project No. 1710, Task No. 171007

2

(Prepared under Contract No. AF 33(616)-7795
by Paul I. Jacobs of
Educational Testing Service, Princeton, N. J.)

FOREWORD

This work was performed by Paul I. Jacobs, Educational Testing Service, Princeton, New Jersey, under Contract No. AF33(616)-7795 with the Aerospace Medical Research Laboratories. The contract was initiated by Dr. Alvin Ugelow, Operator Training Section, Training Research Branch, of the Behavioral Sciences Laboratory. The work was performed in support of Project 1710, "Training, Personnel and Psychological Stress Aspects of Bioastronautics," Task 171007, "Automation of Training Systems" and was carried out from January 1961 to December 1961.

The author is grateful to Drs. Robert M. Gagné, Sharad Kulkarni, Milton Maier, James C. Terwilliger, and Alvin Ugelow, all of whom helped to clarify his thinking on the topics covered in this report.

ABSTRACT

Although there are fundamental differences in the objectives of the two activities, the programming of instructional material bears many similarities to the construction of tests. A systematic comparison of problems and procedures reveals important implications for programming from the older field of testing. Theory and experience in test construction can be especially useful in the selection of valid criteria for assessing the effectiveness of a program, the ordering of instructional subject matter, the writing of instructional frames, and the formal evaluation of the program. Adaptive programming implies measurement of both aptitude and achievement in order to assign trainees to appropriate individual sequences of instruction. Possible applications resulting from examination of these and other issues are explored, and necessary further research is suggested.

PUBLICATION REVIEW

Walter F. Grether

WALTER F. GRETHER
Technical Director
Behavioral Sciences Laboratory

4

TABLE OF CONTENTS

5

Section 1. Introduction
Purpose and Plan of This Report

Almost every person in America has had some contact with tests. People are given tests in school to determine what may be expected of them and what they have already learned, they are given tests so that employers can select the best qualified applicants, they are tested for voter registration, for driver's licenses, etc. Because of this first-hand experience in test-taking, most people have some ideas, correct and incorrect, about what tests are, the purpose of using tests, the value of tests, and their limitations.

In recent years psychologists and educators have paid a great deal of attention to printed materials and to devices which seem to closely resemble tests, but which are called auto-instructional programs, teaching machines self-instructional devices, automated tutors, etc., and which are used for quite a different purpose than that for which tests are used. We will call these printed materials auto-instructional programs, or, simply, programs. Most people have probably not had first-hand experience with programs, although they may have read about them in newspapers and magazines.

The purpose of this report is to examine the extent to which what we know about tests can be applied to the development of programs. Because this purpose is a relatively restricted one, no attempt has been made to deal with either testing or programming in a comprehensive way; the emphasis is on applications of testing to programming.

The report is intended primarily for people now engaged in training and programming activities. For the benefit of those readers who are just becoming acquainted with programming, this section and Section 2 provide general background information on testing and programming. Sections 3 and 4, which constitute the main body of the report, deal with how we can use what we know about testing when we set out to construct a program. Section 5 provides a summary of the report.

Tests and Programs: Similarities, Differences, and Relationships

We use tests to help us decide how to sort out or classify people. We may want to sort them into those who will succeed in college and those who will not; those who should be given a driver's license, those who should be given a license subject to certain restrictions, and those who should not be given a license; those who should be given school grades of "A," of "B," of "C," of "D," and of "F"; etc. In general, tests tell us what people do in certain situations (on test questions or "items"), and we may use this information for such purposes as predicting what they will do in other situations.

On the other hand, we use auto-instructional programs to teach people, or, as a psychologist might say, to modify their behavior so that their performance on some class of tasks is different because of having been through the program. We must keep in mind this fundamental difference in reasons for using tests and for using programs: <u>we use tests to measure present behavior so that we can predict future behavior, while we use auto-instructional programs to modify or produce future behavior</u>.

While tests and programs do differ in what they are used for, there are certain similarities and relationships between tests and programs which suggest that it would be worthwhile to look at our accumulated knowledge of testing to see what implications for programming might exist.

To begin with, tests and programs are similar in appearance. Both tests and programs basically consist of sets of questions. The technical term for a test question is an item, while the technical term for a program question is a frame. It is very frequently difficult to tell from inspection whether an individual question is meant to be a test item or an auto-instructional frame. Which of these questions do you think are meant to be items and which do you think are meant to be frames?

6. You have purchased 7 chances in a lottery for a new car. A total of 10,000 chances were sold. What is the possibility that you might win?

6. What is the sum of 7 and 3?

8. A precursor of Vitamin A is a ______________.*

Often a test question is accompanied by some expository material, that is, some information not in the question itself which the examinee will need in order to answer it. An auto-instructional frame may also be accompanied by expository material. It is difficult, therefore, to tell from the presence or absence of expository material whether a question is meant to be a test item or an instructional frame.

While it may be difficult to tell whether an isolated question is meant to be a test item or auto-instructional frame, the context in which the question occurs will probably enable one to make this distinction. An auto-instructional frame will usually be preceded by other frames which "lead up

*All three questions are frames. From Barlow, Calvin, and Glaser, respectively, as reproduced in Rigney and Fry (ref. 111).

to it" and make it easier to answer it. A test item, however, will usually stand by itself; the test constructor will try to avoid having other items in the same test which help the examinee answer a given item.

An additional clue as to whether a given question is meant to be a test item or an auto-instructional frame is this: after the learner responds to an auto-instructional frame he will usually be exposed to the correct answer. In this way he can find out whether the answer he gave was correct, that is, he can receive knowledge of results. After the examinee responds to a test question, he usually does not receive knowledge of results in those tests that are currently available. This does not mean, however, that providing the examinee with knowledge of results would necessarily defeat the measurement purpose of using the test, or that future tests will not provide knowledge of results to the examinee. In fact, Severin (ref. 119), Pask (ref. 104), and Pressey (ref. 106) have suggested that under certain circumstances tests which provide knowledge of results can do a better job of measurement.

Suppose we found ourselves in the improbable situation, after looking at a set of questions, of not being sure whether it constituted a test designed to measure people's behavior or an auto-instructional program designed to modify people's behavior. Could we not just use the set of questions in order to see how it functioned?

Since we define a program as a set of questions which serves to modify behavior, perhaps the simplest way to collect data in order to classify an unknown set of questions as a test or program is to administer the set of questions twice to the same group of people. If people get substantially more questions right the second time than they do the first time, the set of questions has modified behavior and may be called a program. Numerous studies have shown, however, that when people are repeatedly given sets of questions that are already known to be tests (they have already been successfully used to measure and predict behavior), they get more questions right each succeeding time.

A second way we might collect data in order to classify a given set of items as a test or a program is to see whether the earlier questions affect performance on the later questions. If so, then presumably the set of questions constitutes a program. Unfortunately, certain studies have shown that for sets of questions that are already known to be tests, the presence of some questions affects the performance on others.

The two possibilities we have considered above for collecting data in order to be able to classify a set of questions as a test or a program will not work. In each case sets of questions which are designed to be tests and which are useful as tests have certain characteristics which we might expect only programs to have.* Furthermore, tests have these characteristics when they are administered <u>without</u> providing the examinee with knowledge of results (without telling him whether he is right after each response).

---

*We shall see in Section 4 that sets of questions which are designed to be programs and which are useful as programs may have certain characteristics which we might expect only tests to have.

It appears, then, that although a test constructor may intend his set of questions to merely measure behavior, the actual use of his questions will often (if not always) modify behavior also, and the test constructor will find himself with a program on his hands. If he wanted to construct a program, he might, of course, proceed differently than if he wanted to construct a test. To take one obvious procedural difference, in constructing a program he would arrange for the learner to receive knowledge of results following his responses. But questions which are not accompanied by knowledge of results may also have an instructional function, that is, they may also modify behavior. An interesting example of this has recently been reported by Estes (ref. 36, p. 220). He compared the performance of two groups of Ss who were repeatedly given a set of questions under the knowledge of results (L) and no knowledge of results (T) conditions shown below:

Group 1	L		L	T	T
Group 2	L	T	L	T	T

Time ⟶

Note that the two groups differ only in that Group 2 was given the set of questions without knowledge of results one additional time. Yet this additional opportunity, which one would expect to have only a testing function, had also an instructional function: Group 2 showed 78% retention while Group 1 showed only 52% retention.

All of the preceding discussion may be summarized as follows: not only is it difficult to distinguish between tests and programs upon inspection, but it is also difficult to distinguish between them by collecting data. Questions may both measure and modify behavior (at the same time), even if they are intended merely to measure behavior. It is reasonable to expect, therefore, that our knowledge of test construction will be useful in program construction.

An additional similarity between testing and programming suggests that our knowledge of test construction will be useful in program construction. This similarity is in the general steps one takes or should take in the conception, construction, and evaluation of tests and auto-instructional programs. For both tests and programs, these stages may be labelled Specifying Objectives, Determining the Resources Available, Planning and Developing Items (Frames), Pretesting and Revision, Evaluation, and Providing Information to Test (Program) Users.*

In addition to the similarities between testing and programming which have been noted above, there are two general relationships between testing and programming which also suggest that test construction knowledge will be useful to the programmer. It will be helpful to look briefly at one way of

*In Section 2 these stages will be elaborated upon for auto-instructional programming and in Section 3 they will be elaborated upon for testing.

subdividing the category "tests" and one way of subdividing the category "programs" before discussing these relationships.

A very common type of test is the achievement test. The achievement test is used to determine how much has been learned. One use of achievement tests in school settings is to see whether each student has mastered the material he needs in order to master material in higher grades. One may also use achievement tests in school settings to see how much groups of students have learned under different teaching methods. In the first case, one is interested in making a decision about students, and in the second case, one is interested in making a decision about teaching methods. In both cases, the achievement test gives information on how much has been learned.

Achievement tests may also be used in two ways in conjunction with auto-instructional programming. We may want to find out whether a student has mastered the material to which he has been exposed so that we then can put him in a position to utilize what he has mastered, either in further formal instruction or in a job situation. Or we may want to see how well auto-instructional teaching compares with, say, classroom lecture and discussion teaching. Whenever we use auto-instructional programming we are concerned with both of these questions, and the key to answering them lies in the use of an achievement test. For this reason knowledge about achievement testing is important in the construction and evaluation of auto-instructional programming.

We have focused our attention on a particular type of test, the achievement test, and we have consequently seen that there was an important relationship between testing and programming. We will now focus attention on a particular type of auto-instructional program, and will uncover another important relationship between testing and programming.

One way that auto-instructional programming may be more effective than conventional instructional modes (such as traditional classroom teaching) is that with auto-instructional programming all students need not be presented with the same sequence of material. Students differ in how much they initially know of a given subject matter, in how quickly they can acquire new knowledge and skills, and in the extent of their misconceptions. A teacher, who instructs many students simultaneously, may not be able to provide just the right amount and sequence of material for the most .ficient learning of each student. So the teacher may decide (consciously or unconsciously) upon a sequence of material which he hopes will be both adequate for the slower learners to grasp the essentials and interesting enough to keep the faster learners from becoming bored. He is often unsuccessf..; the pace may be too fast for some students and too slow for others; misconceptions may be cleared up for some, while others may become confused. We may expect this to happen not only when one teacher provides instruction for many students simultaneously,

---

*Other ways of subdividing these categories will be mentioned later in this report as the distinctions are needed. Cronbach states "Although some scheme of classifying tests is a convenience, all such divisions are arbitrary. One of the striking trends is the breakdown of traditional division lines" (ref. 21). For a discussion of "types" of programs, see Rigney and Fry (ref. 111).

but also when one textbook or film or auto-instructional program provides instruction for many students simultaneously.

For this reason, many people who develop auto-instructional programs in order to make learning more efficient have been interested in providing each individual learner with a sequence of material tailored to fit his particular needs and abilities, or at least in approximating this condition insofar as limitations in devices, developmental costs, and operational costs allow. The programmer may provide some learners with more frames, with different frames, or with a different ordering of frames than other learners. In general, when the programmer does not provide each learner with an identical sequence of material, we will refer to this as adaptive programming. At this point, the significance of adaptive programming is this: in order to assign different learners to different sequences of materials, we need to first have some information on the basis of which we can classify them. If we wish to use rather different sequences of material for "fast" and "slow" learners, we must first know who the "fast" and "slow" learners are. If we wish to provide supplementary information to correct a misconception, we must first know which students have this misconception. In general, adaptive programming involves getting some information about the learner so that we can give him material especially suited to him. The procedure of getting this information is, of course, a testing procedure. When adaptive programming is used, therefore, there is an additional relationship between testing and programming.

Section 2. Procedures in Programming

Auto-instructional programming is relatively new, and there are still many unanswered questions associated with it.* There is no general agreement, for instance, as to whether the student should be prevented from making wrong responses, or whether the student should be required to make any overt responses. There is no agreement on whether a single program is best for all students. The programmers who appear to represent different "schools" or "philosophies" of programming (e.g., Skinner, Crowder, Pressey) differ in how explicit they are in describing what steps they follow in developing a program. They also differ in how explicit they are about what they do at each of these steps.

For these reasons it is impossible to present a general yet accurate picture either of how programs are written or of how programs should be written. Instead, we will list and discuss the steps in what, at the moment, appears to be an over-complete and idealized procedure for developing a program.

Stating one's objectives has been mentioned as a first step by Glaser (ref. 57), Holland (ref. 70), Skinner (ref. 123), Stolurow (ref. 127), and others. It has not been mentioned by Crowder (ref. 27), in his expositions of his "intrinsic" programming philosophy and technique.

What is meant by a statement of objectives and why is it needed? Several writers have suggested that a statement of objectives should be a set of items that the programmer wants the students to be able to pass. "One might think of these as answers to questions which might appear on a final examination for a given course" (ref. 15, p. 550). When one person commissions another to develop an auto-instructional program, he must, of course, state what he wants the program to accomplish. In order for this statement to be useful, it must be specific.

Statements such as "proficiency in electronic troubleshooting" and "facility with symbolic logic" are not sufficiently specific. The programmer must state just what it is that the person who is proficient in electronic troubleshooting or facile with symbolic logic can do.** Skinner, for example, has analyzed and further specified the "ability to read" as follows:

> "...a child reads or 'shows that he knows how to read' by exhibiting a behavioral repertoire of great complexity. He finds a letter or word in a list on demand; he reads aloud; he finds or identifies objects described in a text; he rephrases sentences; he obeys written instructions; he behaves appropriately to described situations; he reacts emotionally to described events; and so on, in a long list" (ref. 124, p. 383).

---

*Some general sources of information on programmed instruction are Kopstein and Shillestad (ref. 88), Lumsdaine and Glaser (ref. 95), Rigney and Fry (ref. 111), and Stolurow (ref. 127).

**For further discussion of how to specify objectives, see Mager (ref. 97).

If the programmer is developing the program on his own behalf and has previously taught the subject matter, he may feel that the writing down of his objectives is unnecessary, and, if it must be done, it might be done more easily after the program is written. There is no evidence that a teacher who knows what he wants to teach will do a better job of programming the material after writing down his objectives. But certain logical considerations suggest that it may be a good idea for him to do so.

One of these considerations is that the next step in program construction depends on the prior step of stating one's objectives. This step is to find out where the students stand now, that is, to find out how competent they initially are in the directions desired. One way of getting this information is to administer a test of subject matter achievement to them--a test of the proficiency which the program will be designed to develop. As we shall see again in the next section, the programmer can only construct such a test after he first states his instructional objectives.

Once the programmer has decided what is to be taught and has found out what the students already know that is relevant, the next steps are to organize the subject matter, to spell out the relationships among the components of the subject matter, to specify what must be taught before what, and to decide upon a method of programming which is best suited to the particular material. Among the available methods are Skinnerian programming (refs. 123, 125), intrinsic programming (ref. 27) and Ruleg programming (ref. 40). Samples of these and other methods or "styles" of programming are given by Rigney and Fry (ref. 111).

When the programmer has organized the subject matter and decided upon a suitable style of programming, he can then start the actual writing of the program. General suggestions for program writing are offered by Gilbert (ref. 55).*

According to Rigney and Fry (ref. 111), the program writer does not have an easy or routine task. He does not (or should not) merely add detail to an already existing textbook organization, break down the printed matter into one or two sentence segments called frames, and then delete one word at random from each frame and replace it with a blank to be filled in by the learner. Rather, he must first organize the material in a way that seems best, considering what the learner can initially do and what he wants the learner ultimately to be able to do. Then the programmer must devise expository material and questions which are meant to have specific functions in getting the learner where he wants him. Above all, his writing of the program must be sensitive to feedback from the learner at all times. This means that he must repeatedly present the program to learners, in order to find out whether they are, in fact, learning what he wants them to learn.**

---

*Taber, Glaser, and Schaefer (ref. 129) are preparing an extensive treatment of program writing.

**Neither the writings of Crowder (ref. 27) nor of others who have also used intrinsic programming and scrambled book format (e.g., Gorow, ref. 60; Lawson, ref. 90) are very explicit on whether the provisional frames written by the programmer are tried out on students.

Generally, this involves trying out sets of frames on learners as they are written, or perhaps after some editorial revision by subject matter experts or by colleagues of the programmer.

After the programmer tries out his provisional frames, his next step is to revise them on the basis of the information he gets from the tryout. He may revise particular frames because students indicate they cannot understand what is required of them, because students cannot do what is required of them, because the frames do not provide enough background for the student to handle later frames, etc. The programmer then tries out the revised (and probably expanded) set of frames. He may repeat this tryout and revision process several times before he is ready to evaluate the program in a more formal manner.

For the formal evaluation the programmer works with a somewhat larger group of students than before. Ideally, this group is representative in both abilities and motivation of the still larger group for which the program is intended. If the programmer merely wants to know how well the program teaches, he sends the students through it, and then administers an achievement test (a "posttest") to the students. The achievement test, which measures how competent the students are in the area for which the program was designed, will have been developed as a by-product of the steps of specifying objectives and measuring the learners' initial competence. The programmer will then be able to make a statement of the form "When students having these (specified) characteristics go through my program they are then able to get a mean score of ___ with a range of scores of from ___ to ___ on this achievement test."

In general, however, the programmer will be concerned not merely with how well the program teaches, but with how well it teaches relative to how well some alternate presently used method teaches, and relative also to the cost of each of these methods. He may, therefore, directly compare the program with the presently used method.

A final step in evaluation, one which programmers have not yet taken, is to determine how well the program teaches students whose characteristics (such as aptitudes, abilities, etc.) are known to be different from the characteristics of the group of students on whom the program was originally evaluated. One approach to determining this will be discussed in Section 3 under the heading "Providing Information to Test (Program) Users."

Section 3. Steps in Test Construction and Their Implications for Programming

In this section we will see what the steps are in the construction of tests, how these steps are related to the steps in the construction of programs, and what implications we can derive for the construction of programs.

Specifying Objectives

When a test constructor sets out to build a test for his own use, or at the request and for the use of some other person, his first step is to spell out the specific purpose of the test. In Section 1 we saw that tests are used to classify people. The test constructor must, therefore, specify who the people are that he is interested in, and what his purpose is in classifying them.

The basic reason for specifying who the people are that he is interested in is that a test which is useful with one population is, in general, not equally useful with other populations. A test consisting of questions on arithmetic facts (e.g., Three times four equals ___?) may be useful in sorting out "more competent in arithmetic" from "less competent in arithmetic" third graders, but may not be useful in sorting out "more competent in arithmetic" from "less competent in arithmetic" college students. Furthermore, if the test constructor wants to see if an already available test can be used for his purposes, he must look at the population at which the test is aimed and compare it with his own target population.*

When it comes to programming, we would also expect that a program which is useful with one population is, in general, not equally useful with other populations. A program designed for use by high school students might not be appropriate for use by elementary school students because the elementary school students do not have the appropriate background, that is, they do not know the facts, generalizations, concepts, etc., that are needed to benefit from the program. The same program might not be appropriate for use with college students because they might already know the facts, generalizations, and concepts which the program covers.

We do not now have a body of reliable knowledge that we could use in judging how useful a given program is with a population different from the one for which the program was designed (see ref. 120). One approach to this problem is based upon testing considerations, and will be discussed later in this section. The procedure of using different programs to teach the same material to different learners will be discussed in Section 4.

Once the test constructor has chosen the people he wishes to classify, he can turn to what his purpose is in classifying them. If he is developing a selection test, then his purpose is to classify them according to how adequately they would perform in a given situation. With applicants to college, it may be whether they would succeed if admitted to college; with

---

*Thorndike (ref. 131) discusses some considerations in deciding whether to use existing tests or construct new ones.

applicants for drivers' licenses, it may be whether they would be good drivers; with job applicants, whether they would be good machinists, etc. We will refer to the future performances of examinees which the test constructor wishes to predict as criterion behaviors, or sometimes, criteria.

In order for the test constructor to proceed with the development of a selection test he must ultimately specify the criterion behaviors in operational terms. If he is interested in "success in college" he might measure this success by grade-point average. He might measure "good driving" in terms of accident records, and "good machinists" in terms of amount produced. There may, of course, be several alternate ways for the test constructor to operationally specify what he is interested in. He might, for example, choose to measure success in college by disciplinary record rather than by grade-point average, or by both disciplinary record and grade-point average.

If, rather than a selection test, the test constructor is developing an achievement test, he is interested in classifying people according to how much they have learned. In this case his test <u>defines</u> the criterion behavior, and again, the definition must be in operational terms.

The reader will recall that a first step in programming is for the programmer to specify his objectives. This specification should also be in operational terms, that is, the programmer should state exactly what it is that he wants the learners to be able to do. These tasks constitute the criterion behaviors for the program. The nature of test criterion behaviors and program criterion behaviors is essentially the same, and some tasks may indeed serve as both. We may think of the difference between tests and programs in this way: tests are used to <u>predict</u> or <u>define</u> criterion behaviors, and programs are used to <u>modify</u> criterion behaviors.

What, then, do we know about the choice of criteria for tests that we can use in choosing criteria for programs?

To begin with, we know that the choice of criteria has a fundamental importance in selection testing. If a poor selection test is developed, this will be discovered when it fails to predict the criterion, and, if the budget allows, a better test may then be prepared. If, however, poor criteria are selected, there is no opportunity for empirical evaluation--no feedback from the data--to warn the test constructor.

Similarly, if a poor program is developed, this will be discovered when the criterion behavior is examined. If, however, poor criteria are selected, there is no empirical evaluation, no way for the programmer to discover the inadequacy of the criteria. Whether a given criterion is relevant to the programmer's purpose ultimately does not depend on empirical evidence, but rather on a statement by the programmer and/or the person who requests the production of the program as to what he is interested in. Severin (ref. 119), for example, working with a correction procedure and utilizing a Pressey multiple-choice punchboard, was interested in total number of errors made on a set of items (each item could contribute more than one error), while Stephens (ref. 126), also working with a correction procedure and utilizing a Pressey multiple-choice punchboard, was only interested in the number of errors made by subjects on their first attempt at each item.

Given that the choice of criteria is extremely important, how can the test constructor (and the programmer) choose "good" criteria? We come now to a distinction between ultimate and proximate criteria (ref. 92). The ultimate criteria are what the test constructor is really interested in. The ultimate criteria for a scholastic aptitude test might be measures of the extent to which the student attains the goals of the educational institution; the ultimate criteria for a driver's license test might be how safely and courteously the driver manipulates his car in his everyday driving. Ultimate criteria for an auto-instructional program designed to teach good citizenship might be measures of the extent to which the learner uses his opportunities to vote, of how he participates in community affairs, etc. Proximate criteria are what, for any of a variety of reasons, the test constructor is willing to settle for. One reason for using proximate criteria is that it may be too impractical, too costly, to measure the ultimate criteria. While it would be extremely difficult, or perhaps impossible, to measure a driver's everyday driving behavior, it is relatively easy to collapse the frequently encountered driving experiences such as turning, parking, driving in traffic, etc., into a more-or-less standardized five-minute road examination.

Another reason for using proximate criteria is that the ultimate criteria may not be measurable until long after the tester is interested in measuring them, or perhaps they may never be measurable. While the ultimate criterion for an instructional program on civil defense may be whether the learners can perform adequately in a disaster situation if the occasion arises, the occasion may never arise. For this reason the proximate criteria of how well learners do in a simulated disaster situation may be used. While the ultimate criteria for a scholastic aptitude test may be measures of the extent to which the student attains the objectives of the educational institution, the proximate criterion which is likely to be used for convenience is grade-point average.

Still another reason for using proximate criteria rather than ultimate criteria is that the ultimate criterion behavior of each person may be different, and so measures of each person's criterion behavior will not be directly comparable. In such cases, the use of a proximate criterion may provide a relatively standardized set of tasks on which measures of each person's behavior will be directly comparable. We may, for example, want a test which will predict the ultimate criterion of how well a salesman will sell. We know, however, that a salesman's sales record will depend upon what territory he is assigned to as well as upon how good a salesman he is. If we have no good estimates of the sales potentials of different territories, we may use, as proximate criteria, measures of the salesman's behavior in an artificially constructed sales situation. We might ask each salesman questions (e.g., What would you do if the prospect says he wants the product but doesn't think he can afford it?), or we might see what he does when confronted with a stooge as a prospective customer.

We can see in this example the usefulness of proximate criteria; if we want our criteria for a salesman selection test not to be confounded with the territories to which the salesmen are assigned, we can assign them all to the same "territory," that is, ask them the same questions, confront them with the same stooge, etc. At the same time we can see in this example a danger in the use of proximate criteria: the proximate criteria may not be related to the ultimate criteria; how well a salesman answers

questions about what he would do in certain situations and what he does when confronted with a stooge may not be related to how well he can sell.

It seems that both the test constructor and the programmer face a dilemma in choosing criteria for their tests and programs; the ultimate criteria are what they are really interested in, but it may be impossible in practice to obtain measures of them. The proximate criteria may be convenient and relatively inexpensive to measure, but they may or may not be related to the ultimate criteria.

How might this dilemma be resolved? In any situation in which measurement of the ultimate criteria is not possible, one is not forced to choose between ultimate and proximate criteria but rather one may choose from among different sets of proximate criteria, any set of which may have particular advantages and disadvantages. In choosing a set of proximate criteria we should try to choose a set which we know is related to the ultimate criteria. When it is impossible to find out whether the proximate and ultimate criteria are correlated, we might see whether the proximate criteria are correlated with other proximate criteria. If we cannot do this, then we must satisfy ourselves that the proximate criteria we choose to work with are logically related to the ultimate criteria, which is just another way of saying that the proximate criteria should appear to be related to the ultimate criteria.

Several instances have been reported in programming in which the proximate criteria have not reflected the ultimate criteria of amount learned. Stephens (ref. 126) used the number of errors during training as a proximate criterion. He found that changing the order of frames and of multiple-choice alternatives within frames produced more errors during training, but made no difference on a posttest. Fry (ref. 49) also used errors during training as a proximate criterion. For one group of learners he terminated training on a list of paired-associates after two consecutive errorless runs through the list. A posttest (used as ultimate criterion) showed that this group learned no more than a group given five minutes of training during which no member of the group made two consecutive errorless runs through the list.

Gagné and Dick (ref. 52, p. 40) also found that a proximate criterion of errors during training did not reflect their ultimate criterion of transfer:

> "Regardless of the internal criterion measures which were employed (number of errors, time to learn), the transfer test scores make one reluctant to state that the learning program has truly taught 'equation-solving'."

In the above instances the programmers were fortunate in being able to collect both proximate and ultimate criterion data, and in this way see the inadequacy of the proximate criteria. But what can the programmer do to insure a more adequate proximate criterion in situations where ultimate criterion data cannot be collected?

One point to remember is to avoid what Brogden and Taylor (ref. 10) call the <u>error of illation</u>. One commits this error when one fails to distinguish between direct and inferential evidences of the achievement

in which one is interested. Brogden and Taylor cite as an example of the error of illation the rating of the carpenter's "skillful movements" rather then the products he turns out. In programming, the evidence cited above (ref. 49, 52, 126) suggests that under some circumstances measures of learning during training may provide rather poor inferential evidence of actual achievement.

In addition to the error of illation, there are various kinds of bias in criterion selection which both the test constructor and the programmer should avoid (ref. 10). One is called criterion deficiency--ignoring some aspect of behavior in which one is actually interested. The programmer seeks to make learning more efficient, so it is generally essential for him to measure both amount learned and time taken to learn, and possibly some other things. When he finds that program A produces more learning than program B, he also wants to know how the two programs compare in amount of time taken by the learners. Goldbeck (ref. 58), for example, found that a comparison of three versions of a program with regard to amount learned, led to different conclusions than did a comparison of the same three versions with regard to amount learned per unit time.

Nachman and Opochinsky (ref. 102) found that the variable of class size made a difference in amount learned in class (classes containing fewer students learned more), but that the students in the larger classes apparently studied more on their own time to compensate for this difference. If one were merely interested in amount learned, one might conclude that students would wind up with the same amount of knowledge whether they were in small or large classes. If, however, one is also interested in time taken by students to learn, both within and outside of the formal classroom situation, one would conclude that the smaller class size led to reduced learning time and, hence, greater learning efficiency. In many educational, industrial, and military training situations, in which trainees' time for independent study outside the formal training situation is limited, it may be important to know how much time needs to be devoted to study in conjunction with an auto-instructional program. Failure to consider this would result in criterion deficiency.

Another type of criterion bias is criterion contamination--when irrelevant considerations enter into the measurement of the criterion behavior. The reader may have recognized an opportunity for criterion contamination to occur in an earlier example: when salesmen are assigned to territories which differ in "sales potential," then their volume of sales, as a criterion, will reflect both their selling ability and the sales potential of their assigned territory. At best, if salesmen are randomly assigned to territories, the criterion measure will merely be imprecise; if salesmen are assigned to territories in some systematic way (e.g., on the basis of test scores or performance during training), the criterion measure will be biased.

A classical example of criterion contamination in testing occurs when the criterion measure is a rating, and the person doing the rating knows the examinee's score on the predictor test. A supervisor, for example, may know that a particular subordinate obtained a high score on a selection test, and this knowledge might influence the supervisor (consciously or not) to rate the subordinate higher than his on-the-job behavior would

otherwise merit. In such a case the test would appear to be spuriously better than it actually was.

A study by Hughes (ref. 73) illustrates a possible opportunity for criterion contamination to occur in programming. He used both program-trained and "traditionally" trained groups of students, and evaluated the effectiveness of each type of training by means of an essay-type posttest. If the judges who marked the essays were aware of the group from which the writer of each essay came (as seems likely), then criterion contamination may have been introduced. Hughes is not explicit on this point.

Another programming situation in which criterion contamination may occur arises when the programmer is interested both in a measure of learning just after training is completed and in a measure of how well the learning is retained. One design he might use to compare programmed and traditional instruction groups on both learning and retention is shown below as Design (1).

Design (1)

Programmed instruction	Posttest	Retention Test
Traditional instruction	Posttest	Retention Test

Time ⟶

The programmer would randomly assign learners to either programmed or traditional instruction and test them after instruction and again some time later. If he used this design, the collection of posttest data might produce contamination of retention test criterion data. This contamination might come about if the posttest sensitized the learners to the test items which they would again be exposed to on the retention test. As a result of such sensitization, they might discuss and think about the test items during the time between the posttest and the retention test. This extra experience with the items could then be reflected in retention test performance. If the programmer did not intend to give the posttest and retention tests in operational use of the program, the trainees would not get this extra experience during operational conditions, and the evaluation of the program would have presented too favorable a picture of it.

One way for the programmer to avoid criterion contamination in this situation would be for him to use Design (2):

Design (2)

A	Programmed instruction	Posttest	
B	Programmed instruction		Retention test
C	Traditional instruction	Posttest	
D	Traditional instruction		Retention test

Time ⟶

Design (2) differs from Design (1) in that in Design (2), after training, each group is randomly subdivided into two subgroups; one of these subgroups receives a posttest, and the other subgroup receives a retention

test. The programmer would compare Group A with Group C in order to determine the relative merits of programmed and traditional instruction for learning, and Group B with Group D in order to determine their relative merits for retention.

Another way for the programmer to avoid criterion contamination in this situation would be to use an alternate form of the posttest as a retention test. The study of Gagné and Dick (ref. 52) illustrates the use of alternate forms of a test in conjunction with a program. For further discussion of "parallel tests" and "equivalent tests" see Gulliksen (ref. 65) and Thorndike (ref. 132), for a discussion of "randomly parallel" tests, see Lord (ref. 93).

In addition to criterion deficiency and criterion contamination. a third type of bias is criterion scale unit bias. Suppose that one were interested in using the sales records made by salesmen as a criterion for a test designed to select good salesmen. If one merely counted how many sales each salesman made, these criterion data might not be too meaningful. This is because each sale would not be of equal value to the company employing the salesmen. On the other hand, the total volume of sales made by each salesman or, better still, the total profit to the company in the sales made by each salesman would provide more meaningful criterion data. Such data would be more meaningful because the company is not ultimately interested in the number of sales made by each salesman, but rather in the profit each salesman produces. It does not care whether Salesman A made more sales this year than he did last year, but rather whether his sales resulted in more profit this year than last. It does not care whether Salesman B made more sales than Salesman C this year, but rather whether Salesman B's sales resulted in more profit than did Salesman C's sales. Any one sale is not necessarily as equally valuable to the company as any one other sale, so the use of the number of sales made by the salesman as a criterion measure would result in what the test constructor would call scale unit bias. Since any dollar of profit is as equally valuable to the company as any other dollar of profit, the use of the amount of profit produced by each salesman's sales would provide a criterion in which each unit (dollar of profit) produced by a salesman was just as important to the company as any other unit produced either by the same or by a different salesman (see Brogden and Taylor, ref. 11).

In many situations the programmer may also find the dollar criterion will be useful in providing him with an equal unit criterion scale. If, for example, he wanted to develop a program which would train people to be good salesmen, he would use total profit in the sales made by each trainee (not number of sales made) as an equal unit criterion measure. In some situations, however, the programmer (and the test constructor) may not find it easy to use a meaningful equal unit scale such as money.* Consider a case in which his only available proximate criterion is the number of items right on a 50-item achievement test. The programmer could consider this measure to be on a meaningful equal unit scale if it were equally important to him for Trainee A to get question 1 right

*Some of the complexities of this problem of units in learning situations are discussed by DuBois (ref. 30).

as for Trainee B to get it right, and for Trainee A to get question 1 right as for Trainee A to get question 2 right, etc. In one programming study, Jones (ref. 80) apparently felt that a gain from pre- to posttest of 10% of the group on one item was equally important to him as a gain from pre- to posttest of 10% of the group on another item. In many cases, however, the programmer may have considerable difficulty in deciding whether gains on different items or by different learners are equally important to him. One meaningful basis he might use for such decisions is how costly it is to bring about gains on different items or for different people by the best available alternative training method. This information, unfortunately, may seldom be available.

We have seen that in choosing a criterion measure for either a test or a program there are several types of bias to be avoided. When one is confident he has avoided these, and has a measure or measures which he is interested in, he is ready to consider the reliability of his measures. The reliability of a test measure refers to the consistency with which it yields results. This consistency may be over time, as when the test on different occasions yields similar results; or alternate-form consistency, as when different versions of a test yield similar results; or internal consistency, as when the component items of a test yield similar results.*

If our criterion measure is a rating, as when a supervisor judges the quality of a worker's performance, we would want the rating to be the same whether it is made at 8 a.m. or at 4 p.m., and perhaps whether it is made this month or next month. This would be consistency over time, or "test-retest" reliability. We would also want the rating to be the same even if the worker had had a different supervisor to rate him. Similarly, if a total score on a test is a criterion measure, neither would we want the total score to vary greatly depending on when it is given, nor, if alternate forms of the test were available, would we want the total score to vary greatly depending on what form of the test is taken. For a discussion of factors influencing the reliability of a test, see Thorndike (ref. 131); for a discussion of the reliability of performance tests of an essentially nonverbal nature, see Ryans and Frederiksen (ref. 116).

We have seen that the test constructor (and the programmer) must concern himself with questions of relevance, possible bias, and reliability of criteria. In the happy event that he finds himself with more than one relevant, bias-free, and reliable criterion, how might he proceed?

In some cases criteria of the same general nature may correlate rather highly. In these cases one may choose to work with one of several possible measures on the basis of convenience or economy. French (ref. 47), for example, found that average freshman grades in college were highly correlated with average four-year grades in college. This meant that average freshman grades, which become available three years

*This is a rather simplified view of reliability. For further discussion, see Gulliksen (ref. 65) and Thorndike (ref. 132).

before average four-year grades, could be used as criteria for new tests or scholastic aptitude.

In the above example, average freshman grades and average four-year grades were of the same general nature, that is, in each case the measure was based upon grades assigned by instructors. When, however, criteria are not of the same general nature, they may not correlate highly at all. Consider the dissimilar criteria of speed of performance and quality of performance. The fastest workers may not, of course, do the best work. How, then, can the test constructor, who attempts to predict performance, and the programmer, who attempts to produce performance, deal with both speed and quality of performance?

Brogden and Taylor (ref. 11, p. 141) suggest that the cost accounting principle may be useful in combining dissimilar sub-criterion measures into a single criterion measure. "A tracing out of the exact nature and importance of the effect of each sub-criterion variable on the efficiency of the organization is the essential step which differentiates the dollar criterion from the more conventional techniques." Once again the dollar is suggested as a meaningful unit. Not only may it provide the programmer with an equal unit scale, but it may also permit him to compare and combine measures (such as time and quality of performance) which do not appear to otherwise be comparable.

How might the dollar criterion be used? In an industrial setting employee time can be given a dollar value in terms of wages, benefits, equipment costs, overhead, etc. If a product is being produced, it too can be given a dollar value in terms of the margin of profit in its sale. Then, when a faster worker also is a less accurate worker, that is, when he turns out more units in a given time but they are of inferior quality to those of other workers, we can combine these two different aspects of his performance into a single measure of cost which will be directly comparable with the single measures of cost of other workers. For a discussion of this procedure with numerical examples, see Brogden and Taylor (ref. 11).

As we saw in the discussion of obtaining meaningful equal unit criterion scales, sometimes the programmer will find it harder to apply the dollar criterion than other times. The dollar criterion will not be so easily applicable within the industrial setting when what the worker does cannot be directly related to a tangible product, or again, in educational and military settings. As Cronbach and Gleser put it: "The assignment of values to outcomes is the Achilles heel of decision theory" (ref. 23, p. 109). They point out, however, that any procedure for evaluating outcomes and making decisions involves this assignment of values, and so it may well be desirable to make this assignment explicit to one's self and to others.

Determining the Resources Available

The test constructor who has spelled out his objectives and has developed criterion measures is in a position to assess the resources available to him for test construction. If he is developing a selection test, he will first want to consider how well he can presently predict the behavior of interest to him, and then consider how this prediction can be improved upon. He may already know, for example, that good salesmen have above average verbal ability. He may wonder what personality traits can be used to characterize successful and unsuccessful salesmen. He may proceed to first-hand observations of salesmen's behavior, he may talk to salesmen and their supervisors to find out what they feel leads to success and failure in selling, and he may look at existing records or collect new information on the characteristics of successful and unsuccessful salesmen. For a more detailed discussion of job analysis procedures, see Thorndike (ref. 131, pp. 12-31).

The programmer faces a somewhat different problem at this stage. Since he wants to modify behavior, not predict it, at this stage he wants to assess how close his trainees now are to having the terminal (criterion) behavior. Carr (ref. 15, pp. 557-558) has said,

> "The programmer must also specify precisely the initial S-R connections, i.e., those connections already in the learner's repertory which approximate the terminal S-R connections and from which the transitional S-R connections are to be developed....To the writer's knowledge, no research has been done on the problem of specifying the initial S-R connections on which the program is to be built."

The problem is this: of the large number of S-R connections which the learners possess at the beginning of training, which ones are relevant to the programmer, that is, which ones need to be built upon to produce the criterion behavior? For example, in the programming of automobile driving, the learners' initial knowledge of French and of architecture may be obviously irrelevant, and their knowledge of traffic laws obviously relevant. But what about their knowledge of how the car's engine works, of how to use the meters on the dashboard, and of their verbal knowledge of the relationship between driving speed and stopping distance? In the latter cases it may not be so obvious whether these behaviors are relevant to the criterion, and, if so, what other behaviors are to be built upon them. One approach to this problem will be discussed in Section 4.

Both the test constructor and the programmer face certain limitations in their respective efforts to predict and to modify behavior. These limitations can be grouped into (a) limitations during test or program development, and (b) limitations during operational use of the test or program.

(a) Limitations During Development

The limitations the test constructor or programmer will usually encounter during development are limitations of time, personnel, and

equipment. When a test or program is needed in a hurry, or when the needed personnel or equipment are not available, the test constructor (or programmer) may be forced to do an inadequate job of such things as specifying objectives, pretesting, revision, and the collection of criterion data. As we have already seen, the specification of objectives is an extremely important aspect of test and program construction, and if the test constructor or programmer does an inadequate job here he cannot usually correct this on the basis of later information.

When pretesting and revision are curtailed, the implications for testing may not be as severe as the implications for programming. In testing, if the population of test-takers is large enough and the time available for test-taking long enough, the test constructor can administer more items or tests than will eventually prove useful. He can then ascertain from a sample of the examinees' test papers which are the useful items and tests; from another sample he can check on the usefulness of these items and tests (cross-validation); and then for the remainder of the examinee population he need score only those items and tests which have been found to be useful. In programming, however, the items (frames) are not conceived to be independent, but rather are cumulative; that is, in programming the effects of exposure to individual frames on criterion behavior cannot usually be isolated. Since frame revision on the basis of tryout with students is considered to be a vital part of programming (refs. 69, 86), we may expect that limitations in resources which curtail tryout and revision will seriously affect the usefulness of programmed instruction.

(b) Limitations During Operational Use

Limitations on time available for testing during operational use are also somewhat different from the limitations on time available for students to go through a program. In testing, we conceive of the measurement as taking place at one instant in time, although the actual test administration may last several hours. In programming, we conceive of the instruction as taking place over a finite period of time. When testing time is limited, the test will consist of fewer items, and we may expect the reliability of the test to suffer. When learning time is limited, it is not clear whether the program should consist of fewer frames. Evans, Glaser, and Homme (ref. 39) reported that when more frames were added to a program, the amount of time taken per frame decreased. This presumably reflects the fact that when more frames were used, the "steps" between the frames were made smaller and therefore could be taken more easily and quickly.* Holland (ref. 69) also reports that when a program was revised and lengthened on the basis of student responses during tryout, total time to go through the program was reduced.

If training time is limited so that fewer frames are used and larger steps must be taken, some learners may not be able to take these steps, and after some point in the program they will be unable to benefit from the later frames. If the revision suggests that additional frames are needed, but time limitations prevent their being used, a major advantage of programmed instruction may be lost.

---

*For a thorough discussion of the concept of size of step, see Lumsdaine (ref. 94).

A second possible limitation during operational use of tests and programs is a limitation on supervisory personnel. In testing, when a test is intended to be given by many relatively untrained proctors, the test constructor must make the administration a simple process. In such an instance he might not have separately timed sections of the test, for example. In programming, if no proctor is around and programmed texts are used, it may be possible for the learners to "cheat" by looking ahead at the answers before deciding upon their own answers. We do not know whether this impedes learning; research is needed on this question. The use of machines may prevent cheating but may introduce the need for personnel to maintain the machines. The programmer should anticipate the need for some supervisory personnel with "auto"-instruction.

A third possible limitation during operational use of tests and programs is a limitation in scoring facilities. Tests may be scored by machine, by the examinee himself, or by another person. Traxler (ref. 135) gives detailed considerations in the decision to have machine or human scoring. In programming, the learner's answer is compared with the "correct" answer not for the purpose of scoring but for the purpose of providing him with knowledge of results, that is, for the purpose of telling him whether he was right or not.* This comparison can be made by the device (from programmed textbook to computer-controlled instructional system) if the format of the frame is multiple choice, that is, if the learner is to choose from a specified set of alternatives. If, however, the format of the frame calls for a constructed response which the learner is to compose himself, a problem arises in the comparison of his answer with the correct answer. The nature of this problem will be discussed in a later section on item and frame writing.

Planning and Developing Items (Frames)

When the test constructor has sufficiently specified his objectives and noted what resources are available to him, he is ready to prepare a preliminary version of the test. One of the first considerations will be that of the scope, or extent of coverage, of the test. If the test is intended to predict success in college as measured by grade-point average, should the test include items intended to get at certain personality characteristics (e.g., perseverance, skill in interpersonal relations) as well as at certain cognitive characteristics (e.g., verbal ability, mathematical ability)? The test constructor will attempt to cover those areas which seem important and which his resources allow him to cover.

When a test is intended to measure achievement in an academic area, a procedure is sometimes followed from which we can derive a precaution for programming. This procedure is based upon a distinction made between "subject matter" on the one hand and "ability" or "process" on the other. Ferris (ref. 41), for example, in working out specifications for a new physics achievement test with subject matter experts, considered such

*An additional purpose may be to provide information which can be used to determine what material the learner is to be presented with next. This will be considered further in Section 4.

things as time, mass, geometric optics, and conservation of energy as subject matter topics, and the ability to demonstrate qualitative understanding of fundamentals, to apply knowledge to unfamiliar situations, and to draw valid conclusions from observation and data as abilities. The subject matters and abilities are laid out in a two-way grid or matrix (see Vaughn, ref. 136). The procedure involves specifying the number of achievement test items which are to be developed for each of the intersections of subject matter and ability categories (e.g., geometric optics and ability to apply knowledge to unfamiliar situations). These numbers should reflect the test constructor's relative interest in the various subject matter and ability categories. Such a specification may be useful to the test constructor because it prods him into writing items to cover that in which he is interested, rather than merely writing items for those categories in which item-writing is easy. The specification may also be useful to other people who may wish to use the test, since it communicates the coverage of the test to them.

An important point to remember is that although the test constructor denotes his relative interest in each category by the number of items he assigns to it, the number of items gives no more than a rough indication of the actual contribution of each category to total test score. This contribution will depend not only on the number of items in the category, but also on the standard deviation of the subscores from the category and on the correlation of these subscores with the subscores from the other categories in the test.

How is this related to programming? Evans, Glaser and Homme (ref. 40) have suggested that in setting up specifications for a program, one should also make use of a matrix--a "Ruleg" matrix. Presumably this matrix would also serve the functions of insuring that the programmer cover that which he intends to cover, and communicating to others what it covers, as well as the function Evans _et al_. mention of getting the programmer to interrelate the concepts in the program. An important point for the programmer to remember is that the numbers of frames devoted to each rule may only be roughly proportional to how well each rule is learned. It would seem that different concepts will require different numbers of frames to be thoroughly understood. We already know that in the relatively simple case of learning paired-associates that different pairs require different amounts of practice (e.g., see ref. 85).

Item (Frame) Format Specifications

The test constructor who has spelled out the specifications for his test is ready to consider the item format or formats which he will use. His choice among different item formats may already have been limited if his assessment of resources indicated that the test must be scorable by machine. The choice of formats he makes will probably reflect what, for him, is an unhappy merging of both "practical" considerations, e.g., ease of writing items, ease of scoring, and certain "theoretical" considerations, e.g., "I don't think I can test writing ability with multiple-choice items." While theoretical rationales might be developed for using or for not using any format for any purpose, we do not know whether certain formats are inherently more desirable than others in all situations, or even inherently more desirable than others in a particular situation:

"...any such characteristic differences (in reliability and validity) as may exist among item forms are of trivial consequence when compared with the extreme differences observed among items of the same form." (Ref. 33, p. 189.)

In programming, it appears that characteristic differences among frame formats may also be of trivial consequence when compared to the differences observed among frames of the same format. The available evidence on multiple choice vs. constructed response frame formats (e.g., see refs. 37, 49, 113) does not show outstanding differences between these two formats.* Nor would it be clear what it meant if, say, each of these studies showed the constructed response format to be clearly superior. We cannot specify the relevant dimensions along which multiple-choice and completion formats might differ, so we cannot sample these dimensions to obtain generalizable results.

Writing Items and Frames

For the most part, the suggestions available in the literature for writing test items are based upon informal, uncontrolled observations, "folklore," "common sense" considerations, etc. In one study Dunn and Goldstein (ref. 31) tried to systematically evaluate some of the traditionally accepted rules for writing test items. The rules dealt with "incomplete statement versus question lead, absence or presence of specific determiners or cues to the correct alternative, alternatives of equal length versus extra-long correct alternative, and consistency or inconsistency in grammar between lead and alternatives." Their findings gave no support to any of the four rules with which they worked.

Suggestions for writing auto-instructional frames which are derived from informal, uncontrolled observations, "folklore," "common sense" considerations, etc., are also available (e.g., see refs. 55, 86). These suggestions, like the suggestions for writing test items, are of unknown validity. Furthermore, just as the finding of Dunn and Goldstein had rather negative implications for test item writing, there is a study which has rather negative implications for frame writing. Newman (ref. 103) compared a group of students whose study materials were sequenced and controlled in accordance with principles derived from learning research with a group of students who used their own study techniques, and found that the group using their own study techniques learned more. We do not know, of course, how far we can validly generalize this finding. But the finding should serve to caution programmers against a rigid adherence to insufficiently tested rules for the construction of program frames, just as the

*Frederiksen (ref. 45) has worked with a new response mode in testing which incorporates features of both multiple-choice and constructed response formats. S constructs his answer, then views E's alternatives and "chooses the one which best approximates his response." In this way any advantage of S constructing his response is obtained, while the problem of scoring constructed responses is minimized. Gilbert (in ref. 95, pp. 545-546) has suggested that such a response mode be developed for programming.

study of Dunn and Goldstein should serve to caution test constructors in a similar way. With this background of caution and skepticism, let us look at some suggested rules of unknown validity for writing test items, and see what implications they might have for writing program frames.

Discussions of how to write test items are available in Ebel (ref. 33) and Travers (ref. 134). According to Ebel, the most important suggestion is to "express the item as clearly as possible." In this context "clearly" means unambiguously and understandably. "Test items should not be verbal puzzles. They should indicate whether the student can produce the answer, not whether he can understand the question" (ref. 33, p. 213).

It would seem that, in writing a program it is also important to strive for clear, unambiguous frames. The programmer might use certain available procedures which have been developed to assess the "readability" of his frames. The Flesch count (ref. 44) measures readability by combining measures of average sentence length and number of syllables per word, while the Dale-Chall count (ref. 28) measures readability by combining measures of average sentence length and relative frequency of words not on a list of 3000 easy words. Both counts yield similar results, and the choice between them may be made on the basis of convenience. Dale and Chall point out some limitations in the use of this type of count.* The programmer might use such a count to make his program more readable before he tries it out on students. Research is needed to establish whether this is a feasible way to improve programs. One group of students might be given a first draft of a program, and another group of comparable students given a draft of the program which has been revised on the basis of readability count. The two groups would then be compared on time taken to go through the program and on posttest achievement.

One difference between test items and program frames that we should keep in mind when we try to apply item writing suggestions to frame writing is that in general each test item is self-contained, that is, it must be understood by the examinee when it occurs alone; while each program frame occurs in the context of other frames, and these other frames may serve to clarify its meaning. Consider this frame: "Some errors possible in attempting a(n) ________ response are errors in Content, Language, Depth, and Meaning" (from Ellison _et al._, in ref. 111, p. 99).

If this were a test item, it would not be too clear what is called for. In the context of the program in which it actually occurs, however, the preceding frames serve to clarify its meaning. This example suggests that some of the more specific suggestions for writing tests (for which the empirical basis is not too secure) may not be directly applicable to writing frames.

Let us look at some of these suggestions.

1. "Avoid including two or more ideas in one statement" (ref. 134, p. 56).

---

*An additional limitation for the programmer is that a readability measure may be inappropriate when technical vocabulary is to be taught.

In testing, when an examinee cannot answer an item which deals with several ideas, we have no way of knowing which of the ideas he cannot handle. From this point of view it may be undesirable to use such an item. In one programming study, however, Severin (ref. 119) found that the use of "two pairs" frames, which contained two Russian-English vocabulary word pairs per frame, resulted in more learning than did the use of a two-alternative multiple-choice frame, which contained only one Russian-English vocabulary word pair per frame. In this specific instance, therefore, the suggestion for test item writing does not seem to hold for program frame writing.

2. "Avoid the inclusion of nonfunctional words in the item" (ref. 33, p. 215). Ebel considers a word in a test item to be nonfunctional "when it does not contribute to the basis for choice of a response." Holland makes a similar suggestion for program writing: "It is probably an adequate rule of thumb to say that any portion of an item which is not necessary for the student to arrive at a correct answer cannot safely be assumed to be taught by the item" (ref. 70).

Some suggestions are specific to constructed-response items:

3. "Direct questions are probably preferable to incomplete declarative sentences, especially for younger, less 'test-sophisticated' pupils, because the former are more similar to the forms in which ordinary discourse is carried on.

Faulty: America was discovered in the year ______?

Improved: In what year was America discovered?" (ref. 110, p. 81).

This is one of the four points for which Dunn and Goldstein found no empirical support. We will not consider its implications for writing frames.

4. "Keep the ratio of words given to words omitted very high because, if too many words are omitted, the meaning of the whole will be obscure" (ref. 134, p. 41).

This suggestion would seem to also apply to programming. Below are two examples of frames written by programmers who aim at little or no learner error. In each case the substantial proportion of errors made may be due to the violation of the above suggestion.

"A child has a 'temper tantrum' screaming for candy. The mother gives the child the candy, and the tantrum ceases. The mother's response of handing the candy to the child is ______ by the ______ of the tantrum" (ref. 68, p. 78). Fifty-six percent of all the learners got both answers correct.

"LEARNING is indicated by any 'change' in ______ to a situation which is the result of ______ responses to the same or similar ______, a ______ not nullified to any degree by an extended ______ of ______ during which neither that nor any similar situation is presented" (ref. 5, p. 190). The percentages of all learners who correctly filled in these blanks were 96%, 43%, 86%, 57%, 86%, and 80%, respectively.

The ratio of words given to words omitted may be a rather coarse index of how obscure a frame is. "...one nation, under God, with ____________"

may be an easier-to-complete frame than the frame from Barlow quoted above, yet such an index would rate it as more obscure. Perhaps rather than trying to formulate a rule regarding the ratio of words given to words omitted, we should only conclude that when a frame gives students trouble, the programmer should consider the possibility that too many important words are omitted.

5. "The blanks should refer only to omitted key words" (ref. 134, p. 42).
Holland (ref. 70) compared a group given a program with key words omitted in each frame with a group given a program with "trivial" words omitted in each frame. Sample frames from each of these versions of the program were: A technical term for "reward" is reinforcement. To "reward" an organism with food is to ______ it with food. (key word omitted) A technical term for "reward" is reinforcement. To "reward" an organism with food is to reinforce it with ______. (trivial word omitted) The group with the key words omitted did better on the posttest. Similarly, Jones (ref. 80), working with a multiple-choice format, concludes that the correct answer should not be "trivial." "The good item may be characterized as...one which cannot be answered by reasoning or knowledge of vocabulary alone" (ref. 80, p. 99).

6. "Specify the terms in which the response is to be given.

Faulty: Where is the world's tallest building located?

Improved: In what city is the world's tallest building located?" (ref. 110).

The reasoning behind this suggestion is that it is hard to anticipate all possible answers to a completion item (e.g., "North America" might be an answer to the faulty version of the above item), and so it is useful for the test constructor to state the form the answer is to take (in the above item, the improved version specifies that the name of a city is wanted). With mathematical subject matter it may also be necessary to state the degree of precision wanted in the answer, e.g., the number of significant figures.

In following this suggestion, however, the test constructor is not to choose just any method of specifying the terms in which the response is to be given: "Hints concerning the correct answer, in the form of the first letter of a word, or a number indicating the number of letters in a word, should generally not be employed. Such hints may tend to confuse pupils when the answer upon which they have decided, although it is a correct synonym, does not coincide with the given hint. Guessing and responses to superficial cues may also result from this practice" (ref. 110, p. 82).

In programming, it may be particularly important to specify the terms in which the response is to be given. As we have seen in the discussion of limitations during operational use of the program, the problem of how to tell the learner that he is correct may arise when a constructed response format is used. Programmers have generally left it to the student to compare his response with the "correct" response. This leaves it up to the student to recognize that his response, which may be stated in different terms than the "correct" response, is essentially equivalent to it. This extra burden on the student may be relieved by specifying the terms in which the response is to be made, e.g.,

If A = 1, 2, 3 and B = 4, 5, then A is <u>not equal</u> to __________.

(Use one letter for your answer)

(from ref. 35, p. 15) or by using interchangeable synonyms when providing knowledge of results, e.g., "Latency is the ______ between the onset of an energy change and the onset of a response which it elicits." Time (interval, period) (from ref. 71, p. 4).

Some suggestions are specific to multiple-choice items.

7. If you want to increase (decrease) the difficulty of an item, make the distractors more homogeneous (heterogeneous). Remmers and Gage (ref. 110) give this example: "Which city is nearest to Chicago? (1) Los Angeles, (2) New York, (3) St. Louis, (4) Miami, (less homogeneous); (1) Minneapolis, (2) St. Louis, (3) Cleveland, (4) Milwaukee, (more homogeneous)." The programmer who is interested in a gradual "shaping" of behavior within a multiple-choice format might progressively increase the homogeneity of the alternatives in a series of frames.

8. When it is difficult to anticipate what mistakes will be made in answering an item, do not use "none of these" as the correct answer, since both people who are correct and people who make unanticipated mistakes will choose it (see ref. 33, p. 237). Consider the following frame:

X^2Y^{-3} means the same as:

(A) $X \cdot X - Y \cdot Y \cdot Y$ (B) $(XY)^{-6}$ (C) $\frac{X \cdot X \cdot X}{Y \cdot Y}$ (D) none of these

(from Evans, ref. 38).

The learner who makes any mistake other than (A), (B), or (C), e.g., $(XY)^{-1}$, as well as the learner who knows the correct answer, $(\frac{X \cdot X}{Y \cdot Y \cdot Y})$, will both choose (D). If mistakes other than (A), (B), and (C) are at all common, this might be a poor frame.

9. "Make all distractors plausible and attractive to examinees who lack the information or ability tested by the item"* (ref. 33, p. 234).

Pressey feels that in an instructional test (auto-instructional program), the distractors might be more than just plausible and attractive.

> "Each wrong answer should be one against which a warning is needed, or which elucidates the question in some way. No alternative answer should confuse the student or introduce ways of construing the question which are not educationally profitable to consider. Poor alternatives waste time both in taking the test and in discussion after, and

*Those who reject the multiple-choice format would find the use of this test construction rule in programming to be particularly objectionable: "...effective multiple-choice material must contain plausible wrong responses, which are out of place in the delicate process of 'shaping' behavior because they strengthen unwanted forms " (ref. 123, pp. 140-141). As we saw earlier (page 23), we have no firm basis for favoring either multiple-choice or completion formats in all situations, and it is not clear that we ever will.

might confuse the learner rather than help him" (ref. 106, p. 422).

Pressey's statement suggests that the number of alternatives for a multiple-choice frame should depend on the content of the particular frame, and that all the frames in a program need not have the same number of alternatives. On the other hand, some research in programming (refs. 81, 119) has attempted to compare differing numbers of alternatives as an independent variable. Since this "variable" may actually be a complex of variables (e.g., popularity of alternatives, similarity of alternatives), the results of these studies should be cautiously interpreted.

The suggestions for test item writing given here, which do not exhaust the supply of all possible suggestions, indicate that test construction is a complex, highly skilled activity. This, in turn, might suggest that it be carried out by a professional test constructor. Unless the subject matter is very simple, however, the professional test constructor may emphasize relatively trivial, easily testable aspects of it and neglect its basic structure (ref. 136). Collaboration with a subject matter expert may help to eliminate this danger.

Good programming is also thought to involve both subject matter mastery and programming ability (ref. 111). Because of the relative newness of programming, talent for it may be unavailable, or perhaps unknown to those possessing it.

Rigney and Fry indicate that one skill the programmer must develop is that of going slowly, of proceeding in small steps, "...(the beginning programmer) is quite likely to write the first version of his program with steps that are inappropriate, too difficult, and too few for the material" (ref. 111, p. 14). Other programmers have expressed similar sentiments.

The taxonomy of educational objectives prepared by Bloom et al. (ref. 8) may be helpful in this connection. The taxonomy is based upon six major classes of objectives: Knowledge, Comprehension, Application, Analysis, Synthesis, and Evaluation. As just given, they are assumed to be in hierarchical order, that is, the objectives in one class are "likely to make use of and be built on the behaviors found in the preceding classes in this list" (ref. 8, p. 18).

Bloom et al. present sample items and invite the reader to classify them as to objective, using their taxonomy. This type of task might be useful as a test item in a test used to select programmers. Potential programmers who consistently underestimate the level of objectives, would presumably write items that were too difficult. This type of task might also be useful in training programmers.

Research is needed on the extent to which "experts" agree in classifying items in this taxonomy, and, of course, on whether the classes of objectives are actually hierarchically ordered. We will further discuss the question of ordering behavioral skills in Section 4.

Pretesting (Tryout) and Revision

When the test is constructed, the next step is to pretest it, or to try it out. The test constructor should first try it out on his colleagues, who may offer suggestions concerning format, editorial considerations, ambiguities, and inaccuracies. The term "pretest," however, usually refers to the trying out of the test material on members of the population for which it is intended. Such a pretest may serve several purposes. It may uncover weaknesses in instructions and format, and provide information for establishing time limits, for establishing a desirable test length, and for improving and selecting items.

In programming, several recent reports (refs. 19, 59, 122) indicate that when the learner is not required to make any overt response in going through the program, learning does not suffer and learning time may be decreased. While we do not know whether this finding will hold up with learners not highly motivated by taking part in an experiment, it does suggest that under certain circumstances the learner's overt responses are not necessary during operational use of the program. It would still seem to be highly desirable for the learner to make overt responses during the tryout of the program, however, so that they could serve as a basis for revising the program.

We will now look at various aspects of pretesting and revision in test construction. In each case we will see what implications may be derived for programming.

Instructions and Format

The test constructor may use a small number of people and perhaps a typewritten draft of the test when he attempts to uncover weaknesses in its instructions and format. Conrad (ref. 17) refers to this stage as a "pretryout." During pretryout the instructions may prove to be incomplete, ambiguous, or otherwise deficient.

A pretryout stage seems desirable in program development too. The learner, who may or may not be familiar with some of the more commonly used testing procedures, will almost invariably be unfamiliar with the programming procedure (which will include the novel feature of knowledge of results, and possibly other novel features, such as branching). The programmer's instructions will aim at acquainting the learner with programming procedures, but various misunderstandings on the part of the learner may occur and be revealed by a pretryout.

In addition to format weaknesses in instructions, a pretryout may also uncover weaknesses in how the test was put together. For example, the information needed to answer a question may be on the previous page in the test booklet, or one particular response position may be correct much more than its proportionate share of the time, etc. These weaknesses in putting the test together will be called weaknesses in format.

A test may also be considered weak in format when response sets are allowed to operate. Response sets may be defined as tendencies of subjects to respond in ways which defeat the purpose of the measurement. For example, one response set, "acquiescence," is the tendency to agree with a statement

regardless of its content. The set to gamble is the tendency to guess when the answer is not known. For a discussion of the operation of response sets in personality assessment, see Jackson and Messick (ref. 76).

Since response sets make the interpretation of test scores ambiguous because they measure things the test constructor is not primarily interested in, their influence should be minimized. Response sets tend to occur in situations which are somewhat unstructured and/or too difficult for the examinee. Their influence may therefore be minimized by a restructuring of the test.

Let us look at some response sets which might occur in programming and see what might be done about them.

The learning of paired-associates is a fairly common task with which a program might deal.* In such a task the student must learn to associate particular responses with particular stimuli, e.g., state capitals with names of states, telephone numbers with people, names of symbols with symbols, etc.** If a program always presents the stimulus terms of paired-associate items in the same order, the student may learn a chain of response terms without paying attention to the stimulus terms. This would permit a response set to operate which might lead to the premature termination of training, since the student would appear to be learning the paired associates as paired-associates. The programmer could prevent the formation of this response set by scrambling the order in which the stimulus terms are presented on successive occasions.

In this example, the tendency to learn the response terms as a chain without regard to the stimulus terms may or may not ultimately make it easier to learn the response terms as responses to their respective stimuli. This is a question which might be answered by research on the learning process. The point made here is that the response set in question may interfere with the measurement of the student's proficiency, that is, how well the student does when the stimulus terms remain in a constant order may not be a good predictor of how well he would do if the order were scrambled.

In the learning of "continuous discourse" materials, the programmer may make considerable use of "prompts." In a prompted frame the student is enabled to respond correctly on the basis of knowledge of syntactical restraints, pat verbal associations, etc., for example, "Just as smoke rises, warm air will also ______ " (ref. 94, p. 535). Such prompting techniques are assumed to facilitate learning. It is important, however, to distinguish between frames intended to promote learning and frames intended to see if learning has taken place. The former might be called instructional frames and the latter, criterion frames. The same prompting techniques which may enhance learning on instructional frames should not allow response sets to

---

*In fact, some devices are designed for paired-associate learning material exclusively.

**The associations need not be one-to-one (see ref. 127).

operate on criterion frames. The programmer will detect the more obvious opportunities for response sets to operate by inspecting the criterion frames.

For those frames in which numerical responses are to be given in a multiple-choice format, a special response set may operate. Ebel (ref. 33) notes the "...strong tendency for the examinees to confuse the absolute value of the answer with the response position used to indicate it." This tendency might be reduced (but probably not eliminated) by using letters rather than numbers to indicate the response positions. Shay (ref. 120) reports that some learners showed this tendency in going through his program on Roman numerals, and unfortunately this was not corrected when the program was pretested. Perhaps a color coding of the response alternatives would have eliminated this confusion.

Establishing Time Limits and Length of Test (Program)

The question of how much time to allow for a test is, of course, inseparable from the question of how long the test should be. Administrative considerations usually serve to limit the amount of time available for testing, and in this way indirectly limit the length of the test. For a fixed amount of testing time, the test constructor tries to provide a sufficient number of items to adequately sample the behavior in which he is interested. If, however, he includes too many items, the test may overemphasize speed of responding when the test is intended to measure something else (ref. 135). When time permits and enough items are available, the test constructor may add items to his test to increase the precision of his measurement. Under the proper conditions, the amount of increase in the test's reliability may then be predicted by means of the Spearman-Brown formula (ref. 65).

In programming, we cannot state any precise relationship between length of program and time taken by learners. In the earlier section on limitations during operational use, we saw that adding frames to a program may make each frame easier to respond to, and in this way decrease the amount of time needed to go through the program, while increasing the amount learned. It does not seem reasonable, of course, that adding frames to a program will always decrease the amount of time needed by the learners and increase the amount learned. It may be, however, that whenever frames are added to a program so that the time needed to go through the program decreases, this is always accompanied by an increase in learning. If research supported this hypothesized relationship, it would suggest that during tryout the programmer should pay attention not only to whether frames are responded to correctly but also to how much time is required to respond to each frame. When a frame requires an unusually long time, this might indicate that additional frames are needed prior to it.

An additional timing consideration can be resolved during tryout of a program. There appear to be large individual differences in the amount of time learners take to go through the same program. Rothkopf (ref. 115) reports that a range of times needed on one program is from 23 to 60 hours; Shay (ref. 120), from 31 to 176 minutes; and Gagné and Dick (ref. 52), from 190 to 380 minutes. Since the fastest learners may take only about one-fifth to one-half as much time as the slowest learners, the programmer must make some provision for occupying the time of those who finish first. Tryout data can provide some idea of the range of times to be expected with a particular population of learners using a particular program.

Selecting Items (Frames)

Finally, a major purpose of pretesting a test is to improve and select items. The basic data the test constructor may get from pretesting items are their difficulties and their correlations with other items and with an outside criterion.

Item difficulty is used for two basic purposes. One is to select items, and the other is to order the items selected in the final form of the test. If the test constructor wants to make the maximum number of discriminations among the people in the group tested, and the available items are completely uncorrelated, then he selects items which have p values of .50.* If the items were perfectly correlated, he would choose items spread over the entire range of difficulty. If he wanted merely to divide the group into two subgroups, he would choose items with p values which corresponded to the proportion of people he wanted in each group. If, for example, he wanted the higher scoring subgroup to constitute 20 percent of the population tested, he would choose items with p values of .20. When he uses the item difficulty information for ordering the items in the final form of the test, he generally arranges them from easiest to most difficult.

In programming, frame difficulty information obtained during a tryout may also be very useful in selecting, rewriting, and, perhaps, reordering the frames. Carr (ref. 15) has listed what he considers to be five possible sources of error in program writing. They may be paraphrased as follows: (1) Incorrectly specifying criterion behavior; (2) Incorrectly specifying initially available behavior; (3) Providing an inadequate amount of training material; (4) Improperly sequencing the material; and (5) Moving too quickly. (It is not quite clear how this differs from the third source of error.)

It would seem that each of these sources of error except for the first could be revealed by pretest data on frame difficulty. Unfortunately, it may be hard for the programmer to determine just which source, or sources, of error is operating in a given situation.

A general procedure might be to periodically place in the program what we have called criterion frames--frames which allow the students to demonstrate mastery of some particular aspect of the subject matter. The programmer could obtain data on the difficulty of these frames outside of the context of the program, and compare them with the difficulty of these frames within the context of the program. If a criterion frame is easier within the context of the program than outside it, then the programmer may assume that the frames previous to it in the program contribute toward the learning of what the criterion frame tests for. If the criterion frame is as equally difficult within the context of the program as outside, the frames previous to it in the program may not be adequate.

In addition to item difficulty, the correlations of an item with other items and with an outside criterion are other data which are obtained during tryout and which can be useful in selecting items. We have already seen how

---

*The p value of an item is the proportion of examinees attempting it that answer it correctly.

information about the intercorrelation of the items is used in conjunction with information about item difficulty to maximize the numbers of discriminations made by the test. Now we will see how item-test and item-outside criterion correlations can be used by themselves.

When the test constructor wants his test to measure an ability or trait, e.g., anxiety, then the correlations of each item with total test score become important. Since each item is intended to measure the same characteristic as every other item, the homogeneity of the items, as may be measured by the correlation between each item and total test score,* becomes a basis for item selection. On the other hand, the test constructor might be interested in developing an achievement test to define some criterion behavior in which he is interested, and the correlation of each item with total test score may not be important to him. Finally, he might be interested in predicting an outside criterion, without regard to the "purity" of the test that will best enable him to do so. In this case, he may see how well each individual item discriminates between two groups of people who are high and low on his criterion measure, and then select those items which best make this discrimination. Thorndike (ref. 131, p. 232) points out that the test homogeneity and individual item validity viewpoints are two extremes; in actual practice both considerations may be of some importance to the test constructor.

Can some concepts analogous to those of test homogeneity and individual item validity be useful to the programmer in revising his program?

When one sets out to construct a homogeneous test, then item-test correlations are logically relevant as a basis for item selection. In programming, if we consider each frame to be a test item, there is no logical reason for using item-test correlations as a basis for selecting items (frames). As an empirical matter, however, this may turn out to be a useful procedure. Hosmer (ref. 72) used test items with high item-test correlations as frames for a Crowder-type program. Jones' data (ref. 80), on the other hand, showed a tendency for those items which correlated lower with total test score to have higher instructional value. Jacobs (ref. 77) has discussed some of the difficulties in interpreting Jones' results. We obviously need some research on the usefulness to the programmer of a concept analogous to that of test homogeneity; we will see now one direction such research might take.**

While an individual test item might be judged by the discriminations it makes, an individual program frame may be judged by its instructional effectiveness. The basic paradigm for evaluating an individual frame might be to construct two versions of a program--one including the frame to be evaluated and one omitting it--and to compare the criterion performance of otherwise comparable groups of learners given the two versions. As an operational procedure, however, the application of this paradigm would be extremely impractical with a program of even moderate length. The paradigm might be used

---

*Other measures of test homogeneity, those of Loevinger and Guttman, are discussed by Guilford (ref. 64, p. 363-364).

**In a discussion in Section 4 of the application of Guttman scaling to programming, we will see another direction such research might take.

in a research study to obtain criterion measures of the instructional effectiveness of a set of frames. We could then determine how well item-test correlations could predict these measures of instructional effectiveness.

As an alternative, one might construct a criterion test whose items can be identified in one-to-one fashion with instructional frames or sets of instructional frames in the program. The test would then be administered both as a pre- and posttest. An instructional frame or set of frames which failed to increase the proportion of learners getting the corresponding test item right from pre- to posttest would be revised. Here the problem is whether instructional and test items can actually be matched in a one-to-one fashion without cross-contamination. For a continuous discourse or structured subject matter, that is, one in which certain topics must necessarily precede others, this matching may be impossible.

Revising Items (Frames)

Both the test constructor and the programmer may wish to use the information obtained in a tryout for revising items (frames), as well as for selecting them. In both test construction and program development the successful revision of items or frames may be pretty much of an art, which means that a complete set of rules cannot be explicitly stated for this activity. In test construction only two rules have been found, both of which deal with the revision of multiple-choice items.

(1) Eliminate or revise lternatives which attract very few examinees.

(2) Eliminate or revise alternatives which fail to make the proper discriminations. If the examinees who are highest on the criterion measure choose a particular incorrect alternative more frequently than the examinees who are lowest on the criterion measure, or if they choose the correct alternative less frequently, then the alternative involved is not making the proper discriminations (ref. 131, p. 256).

One might apply both these rules to programming. Rule (1) might not be valid in a programming context, since an alternative which few people choose could still conceivably serve some instructional function by its mere presence among the alternatives. Rule (2) might be rephrased: If high aptitude learners choose a particular incorrect alternative more frequently than low aptitude learners, or if they choose the correct alternative less frequently, then the frame is poor. In such instances there may be a subtle ambiguity in the frame of which only the high aptitude learners are aware.

Further Pretesting and Revision

The amount and kind of pretesting for a test will vary with the available resources. Conrad (ref. 17) recommends a three-stage tryout procedure. The first stage would be intended to reveal gross defects in the test and, as was mentioned earlier, might use the test constructor's colleagues as examinees. The second stage would be for the purpose of item selection and revision and would utilize examinees from the population for which the test is intended. The third stage would provide information on time limits and serve as a "dress rehearsal."

In programming, the amount and kind of pretesting or tryout would also be determined to some extent by the available resources. But in programming

a multiple-stage tryout and revision may have much greater importance than in testing. If the test constructor starts with a sufficiently large number of items, he can discard those which fail to make the desired discriminations and retain those which make the desired discriminations for operational use. In programming, however, when a frame fails to teach what it is intended to teach, it, or perhaps earlier frames, must be revised, replaced, or reordered. The version of the program which emerges from this revision must then be tried out, and this process may have to be repeated many times.

Evaluation

When the final form of a test becomes available after pretesting and revision are completed, the next step is to evaluate it. Since tests are used to classify people, when we evaluate a test we try to find out how well it classifies people. Then, before putting the test into operational use, we compare how well the test classifies people with how well the best available alternate procedure classifies people. Similarly, in evaluating a program, we want to know how well it produces the desired criterion behavior and how well it compares with the best available alternate training method in producing the desired behavior. As we shall see, there are many considerations in evaluating a test which also apply to evaluating a program.

The specific way in which we determine how well a test classifies people depends on our purpose in classifying them. In testing educational achievement, we are interested in what the test constructor calls content validity, that is, "how well the content of the test samples the class of situations or subject matter about which conclusions are to be drawn" (ref. 3, p. 13). Content validity is determined by comparing the content of the test with the content of the instructional or training course and/or the statement of objectives for the course. The test items should not only be derived from the course objectives but also should adequately sample the range of tasks for which the training was intended. A common mistake in preparing a test of educational achievement is to include items which indeed present the examinee with tasks for which the training was intended, but which

> "...limit the test series to the elements of the criterion series that are most conveniently and most easily reproduced, or most easily and objectively observed and evaluated...(so that)...many of the more unmanageable but more important and crucial elements tend to be neglected in, or omitted from, the test" (ref. 92, p. 153).

Since we use an achievement test to evaluate a program, the above considerations concerning the content validity of an achievement test are quite relevant. Many programs may be intended not just to provide the learner with certain "terminal" skills but rather to serve as a basis for learning more advanced subjects. In these cases many problems arise in the proper measuring of "achievement." It may be relatively easy to test what behaviors the learner who has gone through the program can now perform, but this may not be related to how he will learn new material. Kendler (ref. 83) and Gagné (ref. 51) have discussed the problem of measuring how well the learner who has gone through a program can deal with the range of situations in which the programmer is interested. In the terms which we will discuss next, an achievement test which can provide this measurement is said to have predictive validity.

In contrast with content validity, in which performance on the test may be of interest in itself, there are three other types of test validity which are established by relating test scores to criterion scores, namely, predictive, concurrent, and construct validity. Predictive validity refers to how well a test can predict future performance. Concurrent validity refers to how well a test can discriminate among presently identifiable groups. The test constructor who attempts to establish either predictive or concurrent validity faces those problems of collecting criterion data that were mentioned in the earlier section on specifying objectives. Construct validity refers to how well the test measures some trait or quality (construct) which is presumed to be reflected in test performance. The test constructor attempts to establish construct validity by hypothesizing and verifying certain relations between the test and other variables.*

In trying to establish predictive, concurrent, or construct validity, the sample of examinees cannot be as haphazardly assembled as in certain stages of pretesting and tryout. The sample of examinees should be representative in abilities of the population for which the test is intended, and, as far as is possible, representative in motivation as well. This point would seem to apply directly to the evaluation of programs, also.

The test constructor may also be interested in "face validity": whether a test looks like it will do the job for which it is intended.** This concept may also be important to the programmer: if the program does not look like it will do the job for which it is intended, the learners may simply refuse to go through it. We do not yet know what characteristics a program must have in order to possess face validity; for a sampling of some students' reactions, see Roe (ref. 113).

Earlier, in our discussion of insuring the adequacy of a criterion, we mentioned the consideration of criterion reliability, or consistency of criterion measurement. Consistency of measurement is, of course, also desirable in tests which are used for prediction, as well as in tests which are used as criteria, and so test reliability may be looked for in evaluating a test. Thorndike, however, suggests that

> "If anything, the significance of reliability has been overestimated. It must be remembered that precision in a measurement procedure is a necessary condition only for obtaining significant relations between different measures and is not an end in itself" (ref. 131, pp. 104-105).

The programmer may generally want the changes in behavior his program brings about to be lasting rather than temporary, and he might speak of this characteristic as in some way analogous to "test-retest" reliability. Unfortunately, the available knowledge of test reliability does not suggest

---

*For more extensive discussion of construct validity, see Cronbach and Meehl (ref. 24), APA Technical Recommendations (ref. 37).

**For further discussion of face validity, see Mosier (ref. 101).

any techniques for the programmer to use in order to promote the retention of what his program teaches.

Relative Costs and Benefits

We have discussed some considerations in evaluating how well a test classifies people and the implications of these considerations for evaluating how well a program produces the behavior desired. Now we will look at the question of costs of tests and programs and at comparisons with the best available alternatives to tests and programs.

When the test constructor has gathered validity data on his test, he must then reach a decision as to whether it is profitable to put the test into operational use. For a selection test, this decision can be made by considering the validity coefficient of the test (the correlation coefficient between test scores and criterion scores), the relative number of people to be tested and positions to be filled (the selection ratio), and the cost of administering and scoring the test. The higher the validity coefficient, and the lower the cost of the test, the greater the benefit to the test constructor using the test. The selection ratio has a more complicated relation to the benefit obtained through use of the test (see ref. 23, pp. 36-37). When information on these variables becomes available, it can be combined according to formulas given by Cronbach and Gleser. The basis of combining what may be rather diverse measures is a cost analysis. Cost analyses were previously discussed in the section on selecting criteria.

When a measure of the benefits due to the use of the test is computed, it should then be compared with a measure of the benefits due to using the best available alternate selection procedure. The best available alternative may be to use some already available piece of information (e.g., highest grade of school completed, interviewer's impression of applicant, etc.), or it may be merely to randomly select applicants. In any event, the test should be put into operational use to replace or to be used in conjunction with the best available alternative only to the extent that doing so makes a distinct contribution to the test constructor's goals.

In programming, there are also a variety of diverse elements which must be combined to get a measure of the benefits of using a program. We have already seen that there may be several different criterion measures of post-test performance (e.g., rate of performance, quality of performance) which must be combined to yield a single criterion measure for each individual. Criterion measures must further be combined with certain items of cost to arrive at a measure of gain to be expected through the use of the program. Two major items of cost may be learning time and the expense of preparing a program. Ferster and Sapon have stated:

> "...a series of materials could probably be constructed in which each item is scientifically designed so that the students will progress from a zero knowledge of German to a complicated repertory of the level of a year of college German without ever having made an error" (ref. 42, p. 185).

While this may be so, the programmer will want to know such things as how long it would take to go through such a program and how expensive such a program would be to develop.

Rigney and Fry (ref. 111) have outlined the following items which they feel should enter into a cost evaluation:

1. Cost per unit

   a. Per program
   b. Per student
   c. Per machine

2. Investment

   a. Initial
   b. Long term

3. Training time per student

4. Quality of students required (aptitude, experience, etc.)

5. Quality of instructors required (credentials, experience, etc.)

6. Logistics involved

   a. Space, power, maintenance requirements
   b. Program reusability, useful life

7. Practical effectiveness of method

   a. In relation to training objectives
   b. In relation to competing methods

8. Acceptance of method

   a. By students
   b. By instructors
   c. By administrators

An additional cost consideration in many educational, industrial, and military settings is how quickly the subject matter may be expected to become obsolete and how expensive it would be to make changes in the program to cope with this obsolescence.

As in evaluating a test, the basis of combining rather diverse measures in order to evaluate a program is a cost analysis. Kershaw and McKean (ref. 84), although they do not deal explicitly with programmed instruction, present a detailed discussion (with hypothetical examples) of the application of cost accounting procedures to an educational system.* The decision as to

*Such cost analyses, in addition to providing a basis for evaluating programs, may also suggest research designed to reduce costs. Rothkopf (ref. 114), for example, compared two methods of dropping items in the learning of paired-associates, and found that they did not differ in trials to learn or in amount retained per trial to learn, although one method was presumed to involve more expensive equipment.

whether a program should be put into operational use may be made by comparing the net gain to be expected from using the program with the net gain to be expected when the best available alternate training method is used. The best available alternative may be the use of another program, unguided self-study from books, or, perhaps, no training at all. The best alternative most commonly available at present is probably "conventional" or "traditional" instruction (e.g., lectures, recitation classes).

Two major methodological problems arise in comparing a program to be evaluated with the best available alternate training method. One basic paradigm for comparing any two instructional methods is to use one of the methods with one group of students and the other method with a "comparable" group of students and to compare the achievement of the two groups. Conceptually, "comparability" of groups means that the conclusions reached would be the same no matter which group was assigned to which instructional method. Operationally, one tries to obtain comparability by either a random assignment of individuals to methods or by a random assignment of intact groups (classes) to methods. This type of procedure, however, may often not be administratively feasible, and has not always been used in studies of programmed instruction. In some of the pioneering work of Pressey and his students (e.g., refs. 119, 126), as well as more recent work (e.g., ref. 73), the experimenters have resorted to nonrandom assignments. Although they may demonstrate that the groups used did not differ initially on mean aptitude or pretest scores, this may or may not indicate comparability. The evaluation of a program must be based upon experimental comparisons of comparable groups, so that any differences or lack of differences in achievement may be ascribed to differences in programmed and conventional instruction rather than to pre-existing differences in groups.

Another methodological difficulty which comes up in comparing programmed and conventional instruction is this: while programmed instruction may be "standardized" (that is, the material presented to the student does not depend on what class, school, or city he is in), conventional instruction is not standardized. We know that teachers differ markedly in what they do in the classroom, although we know little about teacher differences in the effectiveness of what they do (ref. 100). For this reason, the programmer who wants to compare programmed and conventional instruction must in some way sample the variety of conventional instruction available so that he may have greater confidence that the results he finds will apply to his particular situation. Some programmers and research workers have raised the question as to whether the same program which is useful with "dull" students is also useful with "bright" students. The programmer should also be interested in knowing whether the same program which is useful when "poor" conventional instruction is available is also useful when "good" conventional instruction is available.

Providing Information to Test (Program) Users

In order to use test scores, one needs to know, of course, how the test scores are distributed among the members of a relevant population of examinees. The test user who wants to compare an individual's or group's test scores with those of other individuals or groups can often make a more useful comparison by referring to score distributions from rather specific reference groups. A high school principal, for example, may find it more

useful to know how the achievement of his ninth-grade students on a "standardized" mathematics test compares with the achievement of other ninth-grade students in cities with populations of less than 25,000 in the Mid-west, rather than how it compares with a reference group randomly drawn from around the country. For this reason, the test constructor may make available score distributions for various sub-populations. Similarly, the potential user of a program may find it more useful if the learning times and posttest scores are broken down for various sub-populations.

A second way in which the test constructor might present data, so as to make his test more useful to others, is to establish the equivalence of scores from his new test with scores from other tests of similar use. For example, a potential test user may know that 60 is a useful cutting score for his purposes when the test given is Mathematical Aptitude Test A. If, however, Mathematical Aptitude Test B is the test from which he is given applicants' scores, how can he tell what will be a useful cutting score?

This type of problem arises in institutional testing programs, such as are carried out by the College Board, in which the test items used on successive forms of the tests must be continually changed. The problem differs from the one of providing detailed data on the performance of sub-populations on a particular test in that here one has to deal with a new test given to a new (and potentially different) population.

A basic mechanism in determining or producing "equivalent" scores is to have some overlapping items common to both of two forms to be equated, so that both examinee populations have a common "core" of items. Dyer and King (ref. 32, pp. 101-104) give more details on this procedure as it is carried out by the College Board, and Flanagan (ref. 43) also discusses a number of ways of obtaining comparable or equivalent scores.

The potential user of a program will often find that his intended population is not the same as the population used in evaluating the program. He may specifically want to know up to what level of proficiency the program will bring his learners and how long it will take them to complete the program. The technique of equating test forms discussed above may suggest a procedure for estimating the values of these two variables. While a basic mechanism in test equating is to provide some overlap in the test items given to the two examinee populations, a useful analog of this technique in programming might be to obtain scores for the potential learner population and for the population used in evaluating the program on the same tests, namely, those tests which predict time to learn and posttest scores in the latter population.

Can such tests be found? Carr has stated:

> "One might hypothesize that effective instructional devices might wipe out differences in achievement measures associated with intelligence or aptitude test performance. The findings of a number of experiments seem to support this hypothesis" (ref. 15, p. 561).

He goes on to cite the studies of Porter (ref. 105), Irion and Briggs (ref. 75), and Ferster and Sapon (ref. 42).

The evidence from the studies Carr cites is not too clear-cut. In Ferster and Sapon's study, for example, only six out of the 28 students who started the course, finished it. It may be that the aptitude test which could not predict the ordering of the achievement test scores of the six, could have predicted which students would drop out. Furthermore, the studies of numerous other investigators (e.g., refs. 20, 59, 82, 99, 120) have since shown that aptitude and pretest measures could be used to predict time to go through a program and/or achievement on a posttest. The correlations reported have generally ranged between .30 and .50. Since the potential user of a program may be interested in predicting group means of time to learn and posttest scores, such correlations may indeed be adequate for his purpose.

The success of tne proposed procedure will depend on the extent to which the basic assumption of homogeneity of regression is met, that is, the extent to which an aptitude or pretest measure, which correlates with achievement or time to go through the program in the evaluation group of learners, shows the same correlation in the new group of learners in which the potential program user is interested.

Section 4. Some Selected Relationships Between Testing and Programming

In this section we will deal with two topics which do not usually get much attention in testing but which are quite important in programming: the ordering of frames within a program and the assigning of different learners to different sequences of material. We will explore the extent to which testing considerations may prove useful for each of these topics.

Item (Frame) Ordering

In testing, the ordering of the items is considered only for "motivational" purposes. In general, the test constructor tries to arrange the items in ascending order of difficulty, that is, from easiest to hardest.* If the reverse order is followed, a lower test score may result (ref. 96). While the test constructor may be concerned with how the items are ordered and with the general difficulty level of the items around a given item (ref. 67), these do not appear to be "cognitive" variables, that is, an ascending order of difficulty may facilitate getting the harder items right, but without giving any aids to answering specific items. This is because the test constructor, in choosing items for the test, has followed the rule that "if an item depends in any way upon the preceding one, neither must reveal the answer to the other" (ref. 7, p. 63).

In programming, however, it is commonly believed that there should be a hierarchical relationship between each frame and the next: "At each step the programmer must ask 'what behavior must the student have before he can take this step?' A sequence of steps forms a progression from the initially assumed knowledge up to the specified final repertoire. No step should be encountered before the student has mastered everything needed to take it" (ref. 125, p. 164).

There is little evidence available on this point. Gavurin and Donahue (ref. 54) compared a "logical" with a random sequence of frames, but it is not clear in what way the "logical" sequence was "logical," or how other programmers can provide "logical" sequences for their subject matters. Roe (ref. 113, p. 13) mentions the following anecdote concerning the effects of order of frames on learning:

> "One student, who failed to read the instructions at the beginning of the programmed textbook, read down the page instead of from page to page with the result that the sequence of items he saw were numbered: 1, 40, 79, 118, 157; 2, 41, 80, 119, 158; 3, 42, 81, 120, 159; and so on. This student still managed to get a high score on the criterion test."

While the anecdote is certainly amusing, one wonders whether the learner would

---

*The test constructor will, of course, arrange together those items which depend on the same reading passage, diagram, etc., and also group together those items which are in the same format.

have received an even higher score on the criterion test if he had read the instructions. We certainly cannot conclude that the ordering of frames is unimportant.

If we grant that the programmer should ask what behavior a student must have before he can take each step, how can the programmer answer this question? In many cases the programmer can resort to a detailed task-analysis. If the programmer is attempting to teach "dividing fractions," he must obviously be concerned with the subordinate goal of "multiplying fractions." If he is attempting to teach long division, he must first be concerned with teaching addition, subtraction, and multiplication. In such cases the learning of some skills must precede the learning of others because the skills to be learned first are component parts of the skills to be learned later.

In other cases, however, there may be no such part-whole relationships among the subskills, or, if there are some they may not be immediately apparent. Suppose, for example, one is learning to drive a car. Consider the subskills of maneuvering in traffic and parking. Must one of these skills be taught before the other, and, if so, which should be taught first? We will examine whether testing can contribute toward the answering of such questions. But before we can proceed to explore this possibility, it is necessary to know something about a type of measuring instrument called a Guttman scale.*

With many tests, when we are told only that a given person gets 7 of the 10 items correct,** we cannot say which of the 7 items they were. Or, if we are told that each of two people both got 7 items, we cannot say whether they both got the same 7 items right. If, however, the items in the test form a perfect Guttman scale, then we could, when told how many items a person got right, say just which items they were.

What would such a test look like? We can diagram a generalized scheme of the possible different ways in which people could respond to the items in a Guttman scale. For convenience, let us consider a Guttman scale containing only 4 items. In the diagram below we will let a "1" mean that the person gets the items right and a "0" mean that the person gets the item wrong.

---

*For more information on Guttman scaling, see Guttman (ref. 66), Edwards (ref. 34), Torgerson (ref. 133), Riley, Riley & Toby (ref. 112), Green (ref. 61).

**The Guttman scale was developed in the field of attitude measurement in which the terms "get an item right" and "get an item wrong" are replaced by "endorse a statement" and "fail to endorse a statement."

Response Patterns	Items (1)	(2)	(3)	(4)	Total Number Right
A	1	1	1	1	4
B	1	1	1	0	3
C	1	1	0	0	2
D	1	0	0	0	1
E	0	0	0	0	0

The items might conceivably be:

(1) $3 + 7 = ?$

(2) $8 \times 6 = ?$

(3) $130 \div 5 = ?$

(4) $1041 \div 26 = ?$

If items (1), (2), (3) and (4) form a perfect Guttman scale, then a person answering these items must fall into one of the Response Patterns A, B, C, D, or E. If he gets only one item right, it must be item (1); if he gets two items right, they must be items (1) and (2); etc. In general in a perfect Guttman scale one can reconstruct perfectly from a person's total score exactly which items were gotten right.

Now let us return to see how Guttman scaling may be related to a decision as to whether either maneuvering in traffic or parking must precede the other in a training sequence. In most, if not all, training situations the trainees do not start off with absolutely no background, with no partial knowledge of what is to be learned. We saw in the previous chapter that it is the programmer's job in assessing the available resources to find out just what relevant knowledge and abilities the trainees start with. Suppose, then, that a person in charge of training people to drive automobiles tests each of a large group of trainees on their initial ability to maneuver in traffic and on their ability to park. Suppose further that each trainee is scored pass or fail, 1 or 0, on each of these abilities, and that each of the trainees is found to fit into one of the Response Patterns A, B, or C shown below.

Response Patterns	Maneuver In Traffic	Park
A	1	1
B	1	0
C	0	0

We notice that some people can both maneuver in traffic and park (Response Pattern A), some people can neither maneuver in traffice nor park (Response Pattern C), and some people can maneuver in traffic but cannot park (Response Pattern B), but that no one can park without being able to maneuver in traffic. Ability to maneuver in traffic and ability to park, therefore, form a two-item Guttman scale. What can the person in charge of training validly conclude from this?

It may be quite tempting to conclude that in learning how to drive a car, one must learn how to maneuver in traffic before one learns how to park, but it is not legitimate to conclude this. The fact that "maneuvering in traffic" and "parking" may form a Guttman scale might indeed reflect something about the "inherent structure" of the learning-to-drive subject matter. On the other hand, it might merely reflect the prior learning history of the trainees, that is, it might reflect what <u>has</u> preceded what, not what <u>must</u> precede what. It might be that driving instructors always teach how to maneuver in traffic before they teach how to park, and that people of type B are people who discontinued some prior training after learning how to maneuver in traffic but before learning how to park. Or, people of type B might be experienced drivers who are used to diagonal parking, and the test may have called for parallel parking. If either or both of these explanations were correct, it would not necessarily follow that people of type C, who can neither maneuver in traffic nor park, <u>must</u> be taught how to maneuver in traffic before they are taught how to park.

Of what value, then, is information on whether certain subskills form a Guttman scale to the programmer? We have seen that he cannot use such information to prescribe a necessary ordering of a set of tasks which form a Guttman scale for trainees who initially possess none of these skills. Such information <u>can</u> be useful if the programmer wants to arrange training on various subskills into a given sequence and then allow different trainees to enter this sequence at different points. If, for example, a programmer finds that a set of subskills form a Guttman scale, he may sequence training on these subskills according to how they are ordered on the scale. What a trainee can initially do would be represented by a string of 1's followed by a string of 0's, and he would begin training on the subskill represented by the first 0. The potential economic advantage of this procedure would be that each trainee would not waste time in being taught to do what he can already do, while the ordering of training tasks into a single sequence would greatly simplify administrative matters.*

Whether such a procedure would actually pay off would depend on the relative costs of (1) determining the scalability of subskills, (2) determining each trainees' place on the scale, and (3) training time. It would also depend, of course, on whether a set of subskills which scale were found.

---

*If, for example, six subskills formed the scale, and a trainee needed instruction on only three of them, they would be the last three in the training sequence. If the subskills did not form a Guttman scale, they might be the first, fourth and sixth subskills. If the programmer rearranged the training sequence so as to make these subskills consecutive for this trainee, in doing so he may destroy the consecutiveness for another trainee.

On this last point, Schultz and Siegal (ref. 118, p. 142) have recently found that "Check lists for use in evaluating task performance in several related naval job specialties (ratings)...meet the...Guttman scalability requirements." In their work they did not test each person for various subskills, but rather asked his supervisor whether he was "checked out" on each skill. Research is needed to find out if actual performance tests, rather than ratings, would yield the same result; if not, the scalability of the subskills may only be in the perception of the supervisors doing the rating.

Looking ahead to still later developments, we may find some situations in which the subskills do not at first appear to form a Guttman scale, but when the population of trainees is subdivided into two or more subpopulations, the subskills form different Guttman scales for each subpopulation.

Suppose, for example, in one school system the "topics" in French classes were taught in the order; listening, speaking, reading, grammar, writing, and French civilization, and in another school system they were taught in the order; French civilization, grammar, reading, writing, listening, and speaking. If the programmer in charge of increasing the "knowledge of French" of students who come from these two school systems (and who may have ended formal instruction at different stages within each school) tests them on the various subskills, the subskills would not appear to form a Guttman scale. He may therefore feel that in order to avoid teaching students what they already know, he would need to use many different sequences of material. If, however, he analyzed the test data from students coming from each school system separately, he would find for each school system that the subskills did form a Guttman scale. He could then use this information by providing two different sequences of instructional material and permitting students to enter the appropriate sequence at the appropriate point.

In applying Guttman scaling to programming, what units of analysis should the programmer use? It probably would be more profitable to lump together a set of criterion frames which all deal with the same subskill, score each trainee dichotomously "pass" or "fail" on the basis of his responses to the set of frames, and see whether such subskills form a Guttman scale than to attempt to scale individual criterion frames. Using the subskill as the unit of analysis will reduce the data to a more manageable amount and increase the reliability of the measures used.

We started by discussing how testing might help in discovering what training material <u>must</u> precede what other material, in the sense of being necessary for the learner to benefit from the later material. We have shifted our emphasis to the question of how testing might help in determining what training material <u>might best</u> precede what other material, in the sense of increasing the efficiency of learning.

If there are a set of distinct subskills to be taught, and, by their nature, each of them could precede each of the others, the direct approach to determining the best way to sequence them* would be to try out all

*One might object to the notion that there is a single best ordering for all trainees. We will disregard this complication here.

possible orderings of the subskills with comparable groups of learners. With as few as six subskills, however, there are 720 possible orderings, and Gagné and Dick (ref. 52) have isolated as many as 21 subskills in the relatively limited major skill of solving simple linear equations. It appears, then, that the direct approach to the optimal ordering of subskills will not usually be feasible.

Jones (ref. 79) has worked on the use of simplex theory as a more feasible approach to the problem. Basically he postulates that the pattern of intercorrelations among scores obtained on subskills can suggest how training on these subskills should be sequenced. Jones cites one instance (pp. 90-91) in which he feels this approach paid off. Rao (ref. 108, p. 252), however, suggests that in such a procedure "...the particular conclusions reached for the best sequence of such a training program may well be drawn from actual experience with the problem at hand and not from the loose theory offered." There appears to be a need for an explicit statement of how the theory should be coordinated with observed events, as well as a demonstration of its alleged utility in determining the optimal sequencing of training on subskills. Specifically, there is a need for:

(1) Evidence in a wide variety of training situations that simplicial forms result from the particular abilities needed for various subskills, rather than from the order in which the subskills are taught when data are collected.

(2) Evidence in a wide variety of training situations that changes in the ordering of subskill training which are suggested by simplicial analysis result in more efficient learning. In discussing the instance in which a recommendation based upon simplex considerations was carried out and in which evaluation showed "generally facilitative" effects, Jones (ref. 79, p. 91) states:

> "In this particular instance, therefore, simplicial analysis would have recommended the same course of action without either the expense or the delay of an experimental study. If this result can be generalized, even if only a little bit, the uses of simplex theory in curricula development are very real indeed."

By "generalized, even if only a little bit" Jones seems to mean "generalized, sometimes validly and sometimes not." Unfortunately, unless one can accurately predict when recommendations from a simplicial analysis will or will not be valid, such recommendations will remain of unknown usefulness.

Adaptive Programming

In the previous section on item ordering we saw that under certain circumstances Guttman scaling could be quite useful to the programmer. These circumstances included the condition that different learners could enter the training sequence at different points. The purpose of this was to take advantage of the individual differences that initially exist among the learners; they should not be taught what they already know. In this section we will explore other ways in which the programmer might take advantage of individual differences among the learners, and see what testing considerations might be relevant.

In general, the programmer may try to capitalize on individual differences among the learners by not presenting all learners with the same sequence of instructional material, but rather by giving different learners different sequences of material which he feels are especially appropriate for them. In order to do this he must somehow make distinctions among the learners, either before or during training, or both. We will refer to the procedure of differentiating among the learners in order to assign them to different sequences of material as "adaptive programming." As we shall see, there has been some work done in the testing area on "adaptive testing," that is, testing in which the examinees are provided with different sequences of test items on the basis of their responses to prior test items. It is not, however, because of this work that testing considerations are relevant to adaptive programming. Rather, it is because in adaptive programming the test programmer must make some measurement of the learners in order to assign them to different sequences of instructional material. If measurement is made, then measurement (testing) considerations apply.

Perhaps the most important testing considerations that apply to adaptive programming relate to validity. As we examine various types of adaptive programming we will ask in each case how the costs and benefits of using adaptive programming compare with the costs and benefits of not using adaptive programming.

While the purpose of using adaptive programming rather than linear programming (in which every learner gets the same sequence of material) is to increase training efficiency, this purpose may not always be realized. We have already seen that whether Guttman scaling information is useful will depend on the relative costs of differentiating learners and of instructional time. We must also be concerned with the validity of the test used to differentiate learners. When the test lacks sufficient validity, we cannot expect adaptive programming to pay off. Cronbach has said "The person who attempts to differentiate individuals on inadequate data introduces error even when the inferences have validity greater than chance" (ref. 22, p. 181).

"Recognizing an optimum degree of differentiation makes it necessary to re-examine and qualify statements commonly made in training teachers, to the effect that every pupil has his own pattern and the teacher must fit methods to that pattern, not treat the pupil in terms of the statistical average. ...the teacher who is poorly informed regarding the unique patterns of his pupils should probably treat them by a standard pattern of instruction, carefully fitted to the typical pupil. Modifying plans drastically on the basis of limited diagnostic information may do harm" (ref. 22, p. 183).

While adaptive programming may have potential benefits, the programmer must realize that just as programmed instruction is not necessarily superior to "conventional" instruction, adaptive programming is not necessarily superior to linear programming. With this warning in mind, let us turn to examine some types of adaptive programming.

Fixed-Treatment Placement

The first major type we will consider is that in which on the basis of some pretest, learners are assigned to different but fixed sequences of material. Following Cronbach and Gleser (ref. 23), we will refer to this type of adaptive programming as fixed-treatment placement. In fixed-treatment placement the programmer prepares different fixed sequences of material,

and believes he can identify "types" of learners who will learn most efficiently with each of these sequences.

Under what circumstances would this type of adaptive programming pay off? We can say that it will pay off when the assigning of different learners to different sequences of material results in more efficient learning than would be obtained by assigning all learners to the best single sequence of material. Suppose, for example, the programmer wants to classify all learners as either Type A or Type B, and then give all Type A learners Fixed Sequence of Material I and all Type B learners Fixed Sequence of Material II. It may help the reader to think of Type A and Type B as high and low aptitude people, and Sequences I and II as "large step" and "small step" programs respectively, although the analysis given here will be more generally applicable.

In order to assess whether this adaptive programming pays off, the programmer would first identify who the Type A and Type B people are, then expose randomly chosen subgroups of Type A and Type B people to Sequences I and II. Suppose he found the type of interaction that is shown in Table 1, that is, an interaction in which Type A people learn more efficiently with Sequence I and Type B people learn more efficiently with Sequence II. Then, if the testing and administrative costs were smaller than savings he would achieve, he would give Sequence I to Type A people and Sequence II to Type B people when he put the program into operational use. On the other hand, he might find no interaction, or an interaction such as is shown in Table 2, in which both Type A and Type B people learn more efficiently with Sequence I. In such a case, he would give Sequence I to all people when he put the program into operational use.

In this procedure for determining the payoff from fixed-treatment placement, the programmer must assign half of the learners to a treatment which he thinks is less than optimal. He must keep in mind that the primary purpose of this apparently inefficient procedure is to find out how valid his test is for fixed-treatment placement; it is not to train learners. The test constructor must also use an apparently inefficient procedure in validating a new selection test when he accepts all the applicants: his primary purpose is to validate the test; it is not to discriminate among the applicants.

Can the programmer hope to find interactions of the type shown in Table 1? Stolurow (ref. 127) has summarized much of the experimental literature on human learning and concludes "The studies have provided few specific interaction effects between learner variables and methods variables..."

In a recent study of "adaptive" training procedures Cline, Beals and Seidman (ref. 16) showed that on the basis of aptitude test scores, trainees in a military setting could be assigned to different training sequences aimed at the same goal, with the result being a more efficient operation than if all trainees had been put through the "standard" sequence.

In an auto-instructional setting, however, Shay (ref. 120) found no interaction between three levels of student aptitude and three programs differing in number of frames and "difficulty level." Shay himself points to a number of aspects of his procedure which may have reduced his chances for finding such an interaction. His measures of student aptitude were taken from already available scores on the "...Kuhlmann-Anderson, Detroit Primary, and Public School Primary tests in all but the few cases where recent Binet IQ's were available "(pp. 37-38). He admits "...the possibility that the IQ's for

Table 1

An Interaction Which Is Useful
for Fixed-Treatment Placement[a]

SEQUENCE	TYPE OF PERSON: A	B
I	100	80
II	70	90

Table 2

An Interaction Which Is Not Useful
for Fixed-Treatment Placement[a]

SEQUENCE	TYPE OF PERSON: A	B
I	100	90
II	70	80

[a]Higher numbers denote greater learning efficiency.

the several tests used were not commensurate and may have obscured any real relationship that existed" (p. 59). Another consideration is that because of machine defects, many students were given ambiguous knowledge of results on at least one frame. Finally, as we saw in the last chapter under the topic of pretesting and revision, some students confused the numerical labels. For these reasons*, we should be reluctant to conclude that one could not find the desired interaction using Shay's programs and student populations.

Regardless of what Shay found and how one might choose to interpret his finding, we must, of course, decide in each training situation separately whether it is worthwhile to use a fixed-treatment placement procedure. We may expect this type of programming to pay off when we have some insight into a good choice of tests for differentiating among types of people and among programs to use.

At present, programmers have chiefly, if not exclusively, concerned themselves with measures of general aptitude as a basis for differentiating learners for this type of adaptive programming. Stolurow (ref. 127, p. 59) has pointed out, however, that the "...available research on the relationships between the learner's ability and his gains in learning do not justify the assumption that different programs have to be written for high and low ability groups."

Three general conclusions seem to emerge from the research relating aptitude to learning: (1) Aptitude is positively related to learning; (2) Aptitude is not related to learning; (3) Aptitude is negatively related to learning. Among the possible sources of contradiction in this research are the use of different intelligence measures, the use of different types of learning scores (gain scores, final achievement scores; time per unit scores, units per time scores, number correct scores, etc.), different degrees of experimental control over data collection (E paced; S paced; laboratory setting, school setting), different aptitude measures, and different types of learning tasks (verbal, psychomotor, etc.). Even if the available evidence consistently showed high positive correlations between aptitude and learning measures, this should not lead the programmer to use measures of general aptitude for fixed-treatment placement, since he is primarily interested in the differential payoff from various treatments, rather than in predicting achievement within one treatment. "General mental ability...is likely to be correlated with success in mathematics no matter how the subject is taught. If the alternative teaching procedures are an abstract deductive method and an applied inductive method, the bright students should do better with either approach. ...On the other hand, there may be other qualities of the individual (say, interest in abstract problems, or liking for rigorous reasoning) which would have quite different relations to the two treatments. A measure which predicted success under one treatment and not the other would be a much better aid to placement than a measure which predicts both" (ref. 23, p. 68).

Stolurow (ref. 127, p. 51ff) provides a good discussion of the "qualities of the individual" suggested by the literature on human learning. A number of rather recent studies suggests some additional qualities of this kind.

---

*For a discussion of another, somewhat more technical reason, see Shay (ref. 120, pp. 59-60).

Allison (ref. 2) reports that "Measures of learning and measures of aptitude and achievement, which have generally been treated experimentally as separate entities, have factors in common with each other." The seven interpretable learning factors he found were Verbal Conceptual Learning, Spatial Conceptual Learning, Mechanical-Motor Learning, three Rote Learning factors, and an "Early vs. Late" learning factor. In some cases it may be possible to alter a training situation which involves primarily mechanical-motor learning so as to involve spatial-conceptual learning. Then trainees with high mechanical-motor or spatial-conceptual factor scores could be assigned to the more appropriate version of the training situation.

Bruner (ref. 13) reports that subjects who are provided with material to learn which contains their own preferred type of mediator (thematic, generic, or part-whole) remember better than subjects who are not given their preferred type of mediator.

Messick and Hills (ref. 98) have shown that there are characteristic individual differences in the amount of information needed to make "inductive leaps." This may be important for programs which require the student to induce rules.

Jenkins (ref. 78) found that subjects who give common word associations learn lists of high and medium built-in associations faster than subjects who given uncommon associations, but they learn lists with low built-in associations more slowly.

Finally, we should note the recent work in testing on "moderator" variables (e.g., refs. 46, 48, 117). In this work a preliminary test is used to classify subjects into two or more groups. For each group the further tests to be given can be weighted in such a way so as to provide maximum validity for that group. Frederiksen and Gilbert (ref. 46), for example, first classified engineering students as being either high or low in interest in accounting. They found that a measure of interest in engineering could better predict grades for those who had a low interest in accounting than for those who had a high interest in accounting. Further work in this area might suggest to the programmer what test variables would be useful in fixed-treatment placement.

Branching Programs

Now we turn to the second major type of adaptive programming, the <u>branching program</u>. In this type of program the material presented to the learner is always contingent upon his response to the previous training material. The reader <u>will</u> recall that in fixed-treatment placement the learner is assigned just once to a fixed sequence of frames on the basis of his responses on a pretest; in a branching program, the learner is periodically reassigned during training on the basis of his responses during training. These reassignments may occur as frequently as once per frame.

In the branching program as it has been developed by Crowder, the burden of instruction is placed upon relatively lengthy expository material. This material is then followed by what is essentially a test item, to determine whether the learner has grasped the point of the expository material and can proceed, or has failed to grasp the point and must receive some remedial material. Following any necessary remedial material, the learner is returned to the missed item to attempt it again.

The branching program is sometimes spoken of as involving a "two-way interaction between instructor and student," a "communication process," or a "closed loop," as contrasted with an "open loop" Skinner-type linear program. This language is used to stress the fact that in a branching program not only does the learner get feedback (knowledge of results) from the programmer, but the programmer also gets feedback from the learner. Perhaps this point is overemphasized. In both a branching program and a linear program the programmer gets feedback from the learner: in a Skinner-type linear program the programmer uses the feedback during a tryout stage to minimize errors during training, while in a branching program the programmer uses the feedback during training to determine what material the learner should be exposed to next.

In testing, some work has been done on making the item given to the examinee next depend on how he responds to the preceding item. Hutt (ref. 74) studied the effectiveness of a branching testing technique with the Stanford-Binet. For an "Adaptive" group of Ss, failure on a given item meant that a relatively easy item was given next, so that failure on successive items was rare. Among poorly adjusted Ss, the adaptive group achieved higher scores than a group administered the test under standard conditions.

Krathwohl and Huyser (ref. 89) and Bayroff, Thomas, and Anderson (ref. 6) have also developed branching tests. So far no generalizations that might be useful to the programmer have emerged from this work in adaptive testing.

How can the programmer determine whether a branching program pays off? Once again, the basic procedure is to compare the costs and benefits of using a branching program with the costs and benefits of using the best available alternate procedure, which presumably will be a linear program.

Silberman et al. (ref. 121) have suggested a refinement of this procedure in order to determine if any instructional effect which may be attributed to branching is due to the diagnostic-remedial effect of branching or merely to the extra material a group given a branching program gets. Branching had no diagnostic-remedial effect in the particular program with which they worked.

In another study Coulson and Silberman (20) found no difference between branching and nonbranching groups on a posttest dealing with the elementary psychology subject matter of the program. Just as we cannot conclude from Shay's work that fixed-treatment placement will never be useful, we cannot, of course, assume from this finding that branching will never be useful. Whether a branching program will be successful will obviously depend, among other things, on the programmer's skill in drawing inferences from the learner's wrong responses. In a Crowder-type branching program, the programmer attempts to infer why a particular error was made, so that the underlying misconception or faulty process can be cleared up. In this approach the programmer assumes that the particular errors learners make convey some information, that is, that different wrong responses reflect different processes in the learner. He further assumes that to some extent learners are similar in the misconceptions they initially hold or develop while going through the program, and in the way in which their misconceptions manifest themselves in errors.

Are these assumptions justified? We turn to the testing literature for an answer. From his clinical experience, Rapaport states that "...many

of the intelligence test responses are highly conventionalized; and that a subject knows who was President of the United States before Roosevelt merely adds to his general score. But where the response deviates from the conventional, the deviation does not merely fail to add to his score; it must also be considered as a characteristic which may give us material toward the understanding of the subject" (ref. 109, p. 40).

Davis and Fifer (ref. 29) found that the particular wrong responses made by examinees did convey information; when scoring weights were developed for the misleads in multiple-choice items, the predictive power of the test was increased. The programmer, of course, is not interested in measuring the learner's aptitude from his errors but rather in inferring the processes which lead to his errors. Can the information conveyed by wrong responses be used for this purpose?

Findley and Read (in ref. 9) showed that errors made in answering arithmetic questions may be classified in a way which shows differences in mean total test score among the examinees making errors in different categories. Apparently, then, the errors made by different examinees on different questions can be categorized in a meaningful way. From the nature of the categories (e.g., "interchanging the unknown with whatever lies on the opposite side of the equation," "an error resulting from the confusion of division with subtraction") we may assume that they reflect different processes in the examinees.

Some other studies which also deal with arithmetic errors suggest that the inferences one can draw from individual errors may be rather limited. Grossnickle (refs. 62, 63) found that in general students were not consistent in the types of errors they made in division. In one of his studies only four of the 21 types of errors made were at all persistent.

Brueckner and Elwell (ref. 12) studied errors made in the multiplication of fractions. They found that a student who made an error on one example did not necessarily make errors on similar examples, and, if he did, the errors were often not of the same type. "The pupil should be given at least three or four opportunities to solve examples of one type since single errors may be largely chance or accidental" (p. 177).

In both the Grossnickle and the Brueckner and Elwell studies the experimenters assumed that when a student makes an error on one example and does not make the same type of error on other examples which afford him the opportunity to do so, the error made was a "chance" error. An alternate explanation is that the experimenters were not classifying errors into "types" in the most fruitful way and that some other classification schemes would show that certain types of errors did occur consistently across samples of behavior. In any event, Grossnickle and Brueckner and Elwell presented their problems in formats which may have emphasized rapid, relatively "mechanical" work. Under such circumstances what we may call "chance" errors may result.* We should keep in mind, of course, that the "chance" error category is a residual category for left-over errors which may in the future be better classified in yet-to-be-developed error classification schemes.

*This possibility was pointed out by Leighton Price in a personal communication.

The analysis of data in laboratory studies of human learning has also sometimes involved the classification of errors (e.g., 50). Recently Cook (ref. 18, p. 2) has developed a generalized scheme for classifying errors which occur in paired-associate learning experiments. He makes a basic distinction between a legitimate response (a response which is "... any one of the...response terms in the experiment, whether this response has been elicited by its proper stimulus term or by some other"), an extraneous response (any response other than a legitimate response), and an omission (no response). He then further subdivides the legitimate and extraneous response classes.

The scheme may not be as content-free as it appears. Cook assumes that the set of legitimate responses is part of a larger class of responses. He, therefore, subdivides the extraneous responses into those which are and those which are not members of this larger class. For example, if A, B, C, D, and E constitute the set of legitimate responses, he would consider "Q" to come from the same larger class that the legitimate responses A, B, C, D, and E come from, and he would consider "3" not to come from that class. But one could conceive of the legitimate responses A, B, C, D, and E as coming from the class of the first 13 letters of the alphabet, rather than of the entire alphabet. In that case "Q" would not come from the same class as the legitimate responses come from.

It seems that the experimenter must have some intuitive idea of the larger class to which the legitimate responses "belong" in order to use Cook's scheme. Presumably this intuitive idea would come from his knowledge of the mediational processes common to his learners.

As it now stands, Cook has found the scheme useful in reanalyzing the data of Kopstein and Roshal (ref. 87) and of others. The scheme could be useful to auto-instructional programmers if remedial actions were specified for learners who make responses falling into the various categories. Up to now, adaptive programming in the learning of paired-associates has been limited to the dropping of pairs from a list when the learner's responses indicated they were mastered (e.g., ref. 114).

In general, if different wrong responses do convey information concerning the particular misconceptions held by the learner, how can the programmer use this information to provide the learner with remedial material? The programmer's task is to determine what errors are commonly made, and what misconceptions or faulty thought processes they reflect. In some cases the programmer may be sufficiently familiar with the subject matter and with the characteristics of the learners for him to know what errors are commonly made and what they reflect. In other cases he can determine what errors are commonly made by pretesting his frames in constructed-response format. He would then use the commonly given wrong responses as alternatives in the multiple-choice format of the program. This procedure is sometimes useful to the test constructor (see Adkins and Toops, ref. 1) who selects as multiple-choice alternatives those answers which not only are rather commonly given but also which tend to be given by examinees with lower total test scores. Research is needed to determine whether the mean aptitude or variability of aptitude of those learners choosing a particular multiple-choice alternative is related to the usefulness of providing remedial material for that alternative.

The programmer who has determined what the commonly given wrong responses are must then find out what they reflect. Buswell and John (ref. 14) found

that they could get at the mental processes that children go through in doing arithmetic problems by interviewing them and asking them to "think aloud." Pressey and Campbell (ref. 107) report that interviews were useful in getting at the reasons for spelling errors in the capitalization of words. The programmer may also find interviews useful in finding out what wrong responses reflect.

The programmer who has identified common errors, and gone on to determine their source and to provide remedial material, has done the necessary preliminary work in writing a branching program. He can now go on to the writing, trying out, and revising of branching material.

Measuring Achievement in a Branching Program

The programmer might do well to make branching in his program contingent upon the learner's responses to several frames, rather than to a single frame. This recommendation is in line with Brueckner and Elwell's previously cited finding, and also Gilbert's suggestions (ref. 56). A further rationale for this recommendation is that in most cases it is impossible to adequately sample a domain with a single test item. "An item with a validity coefficient as high as 0.25 or 0.30 usually represents an outstandingly valid item" (ref. 131, p. 245). Finally, Crowder (ref. 26) has recently pointed out that the relatively inexpensive scrambled book can be used even when the programmer wants to make branching contingent upon the learner's responses to several frames rather than to a single frame.

In some existing branching programs (e.g., ref. 25), the learner is given remedial information when he gives a wrong response to a frame and then he is returned to the frame he missed. The frame is now called upon again to serve as a one-item achievement test. Where initially this frame may have been inadequate for this purpose, it is now even more inadequate for this purpose since the learner may be able to remember and reject the previously made incorrect response. This would increase the likelihood that he will choose the correct answer merely by guessing.

In addition to using responses to several frames as a basis for branching, the programmer may use "alternate forms" of the branching frames to help resolve this difficulty. After he receives remedial information, the learner should be presented with frames which cover the same concept as was covered by the previous frames used for branching, but in which some specifics are altered. Again if the programmer uses several alternate form frames he would have a better basis for further branching.

Consider this example: After presenting a definition of "factors," Crowder (ref. 25, p. 14) asks, "Which of the sets of numbers below are the factors of the number 15?

3 and 4	Page 19
3 and 5	Page 25
2 and 13	Page 31."

An alternate form criterion frame which reads as follows might be given to the learner who has turned to page 19 or 31 and received remedial material:

Which of the sets of numbers below are the factors of the number 21?

3 and 7	Page 6
5 and 7	Page 9
7 and 14	Page 10.

The programmer could also use the alternate form frame technique in frame revision. If, following remedial information, the learners do not choose the correct answers on the alternate form frames substantially more often than chance would allow, the programmer should check over his diagnosis of the original error and the remedial information he has provided, since revision is indicated.

Guessing

It has probably occurred to the reader that when the learner in a branching program guesses, the potential advantages of a branching program over a linear program are reduced. The recommendation to use responses to several frames as a basis for branching will help minimize the effects of guessing. A second way the programmer might minimize these effects might be to instruct the learners not to guess. Swineford and Miller (ref. 28) studied the effects of such instructions in a vocabulary testing situation. They found that instructing the subjects not to guess reduced but did not completely eliminate guessing. They measured guessing by the number of times the subjects attempted to provide synonyms for nonsense words.

Their technique of measuring guessing suggests a way to train learners not to guess: Learners could be given nonsense frames for which all alternatives lead to negative knowledge of results ("wrong"). These frames would be interspersed with legitimate frames. The nonreinforcement of guessing on the nonsense frames would be expected to decrease the future occurrence of guessing. Successful guessing on legitimate frames would mean that guessing on frames-in-general would be intermittently reinforced, but this intermittent reinforcement is always present. The nonsense frames would serve to decrease the percentage of times that guessing is reinforced.

The reduction of guessing through instruction or through training can only be effective, of course, if the learner who realizes he has no idea of the correct answer to a frame is given the opportunity to say so. The programmer can provide this opportunity by using "I don't know" as a response alternative. A "don't know" response may also provide useful information during tryout and revision. If it tends to be chosen by learners of higher aptitude, a subtle ambiguity may be present in the frames.

Up to now we have confined ourselves to considering only the particular errors made by learners as a basis for branching. Now we will turn to a theoretical formulation which will lead us to consider the use of response time as a basis for branching.

Response Time and Branching

Amsel (ref. 4) has provided a theoretical framework for identifying situations in which it may be quite desirable to have errors committed during training. When the learner starts out with a strong (superthreshold) correct response tendency and a strong (superthreshold) incorrect response tendency,

the programmer who merely elicits and rewards the correct response may not produce a sufficiently great difference in strength between the correct and incorrect response tendencies. Amsel hypothesizes that in these situations the programmer must also elicit (but not reward) the incorrect response in order to be certain that the incorrect response will not be given after training is complete. It would seem that Amsel's analysis, if valid, would apply more generally to all situations in which there is initially a superthreshold incorrect response tendency, whether or not there is also a superthreshold correct response tendency.

How could a programmer detect these situations? In some cases, a superthreshold incorrect response tendency may obviously be present, e.g., in the context of learning binary arithmetic, the tendency to respond "2" to the frame "One plus one equals __?" In less obvious cases, the programmer might detect such tendencies by pretesting frames in a constructed response format on a population comparable to the learner population. This procedure might be useful in detecting tendencies which were common to many of the learners. The programmer would then write into the program several frames designed to elicit and not reward the commonly given incorrect response.

The programmer could also deal with superthreshold incorrect response tendencies which are idiosyncratic, but this would, of course, require a branching program. It would also require a more flexible teaching device. Specifically, the teaching device would have to be sensitive to the "response latency," that is, the time interval between the presentation of the frame and the learner's response to it. Any response made with a latency shorter than a given value could be considered to indicate a superthreshold tendency. The teaching device might be set up to repeatedly expose the learner to any item on which this occurs but to only repeat once an item on which an incorrect response with a longer latency was made, or on which no response was made.

There is some evidence which suggests that if the above procedure were used, then the critical response latency should be set at different points for different learners. Tate (ref. 130) gave arithmetic reasoning, number series, sentence completion, and spatial relations test items at each of three difficulty levels to a group of subjects. He found that each S had a characteristic speed of response which was relatively independent of the subject matter of the item, the difficulty level of the item for the group, and whether or not he got the item right. Research is needed to explore the diagnostic value of response latency information in general and the usefulness of adjusting the critical latency to the individual.

Finally, in addition to branching on the basis of the particular error made and on the basis of time taken to make an error, there is the possibility of branching at the discretion of the learner. Silberman et al. (ref. 121) found that a branching program was not superior to a fixed (linear) program when the conditions for branching were prescribed by E, but a program in which S had the option of branching was superior to a fixed program. At the moment there appear to be no specific applications of testing considerations to this type of branching.

Section 5. Summary

This report is concerned with the implications of testing for auto-instructional programming.

In Section 1 we saw that tests were used for predicting behavior and programs were used for modifying behavior. We noted that in spite of this difference in purposes, the steps one goes through in developing a test and in developing a program were similar. We further noted two relationships between testing and programming: (1) Tests are used in the evaluation of programs; (2) Tests are used in adaptive programming in which different learners are assigned to different sequences of material, to differentiate among the learners.

In Section 2 we briefly went over the steps in the construction of a program. These steps are Specifying Objectives, Determining the Resources Available, Planning and Developing Frames, Pretesting and Revision, Evaluation, and Providing Information to Program Users.

Section 3 was organized around a more extensive discussion of the steps in the construction of a test and some implications for programming that emerge at each of these steps. In both testing and programming the first step is the specification of objectives, and this ultimately involves a choice of operationally defined criteria. Various considerations that the test constructor takes into account in choosing criteria are relevance, possible bias, and reliability. The importance of these considerations for the programmer was discussed. The point was made that the programmer was apt to look for internal criteria, that is, measures during training of how well the learners were doing, and that such criteria may be rather poor. The necessity of combining criteria and the dollar criteria technique for doing so were both discussed.

The assessment of the available resources was then discussed, and we saw how the significance of this step differed for the test constructor and the programmer. Item writing suggestions were examined with the purpose of seeing how they might apply to programming. We saw that specifying the terms in which a constructed response is to be given, which may facilitate scoring in testing, may be more crucial in programming since it may affect whether the learner gets knowledge of results. It was suggested that the task of classifying test items as to educational objective may be useful in the selection and training of programmers. Some research necessary before implementing this suggestion was pointed out.

We went on to consider the differences in selecting test items and program frames and the greater importance in programming of further pretesting. Under evaluation, we saw a basic similarity of approach: both tests and programs are compared as to costs and benefits with the best available alternate procedure. In order to carry out this comparison, many diverse elements need to be combined, and again the dollar criterion is useful. Finally, both the potential test user and the potential program user need to know how well a test or program developed for use with one population of people will work with a different population. The possible use of a procedure analogous to test equating, for estimating how useful programs will be for different populations, was discussed.

Section 4 was concerned with two programming problems which do not usually occur in testing, but for which testing considerations are relevant.

The first problem concerns the optimal ordering of the instructional material. It was pointed out that Guttman scaling may be useful if the programmer wants to start different learners at different points, in order to capitalize on initial differences in their capabilities, and at the same time keep the same sequence of instruction. Simplex theory may also prove useful in the ordering of instructional materials but at present it needs further development.

The second problem was that of adaptive programming, the providing of different learners with different sequences of material. One type of adaptive programming is what the test constructor would call fixed-treatment selection. Testing considerations may be used in its evaluation. Possible variables which might prove useful in assigning learners to different sequences of material in fixed-treatment placement were pointed out.

Another type of adaptive programming is branching, making the learner's sequence of material depend on his responses during training. The tryout of frames in the completion format may provide the programmer with information as to what errors are commonly made; these can then be used as alternatives for the multiple choice format of the branching program. Interviews may be helpful in finding out what misconceptions or faulty processes are reflected by these errors. Alternate form criterion items may be useful in testing the effectiveness of remedial sequences both during revision and during operational use of the branching program. Making branching contingent upon responses to several frames may increase its effectiveness. Response latency may have diagnostic value in a branching program; some research possibilities based upon Amsel's theoretical views were pointed out.

REFERENCES

1. Adkins, D. C., & Toops, H. A. Simplified formulas for item selection and construction. Psychometrika, 1937, 2, 165-171.

2. Allison, R. B., Jr. Learning parameters and human abilities. ONR Technical Report and Doctoral Dissertation, Princeton Univer., Princeton, N. J.: Educational Testing Service, 1960.

3. American Psychological Association, Technical recommendations for psychological tests and diagnostic techniques. (Supplement to Psychol. Bull., 1954, 51.)

4. Amsel, A. Error responses and reinforcement schedule in self-instructional devices. In A. A. Lumsdaine and R. Glaser (Eds.), Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

5. Barlow, J. A. Conversational chaining in teaching machine programs. Psychol. Rep., 1960, 7, 187-193.

6. Bayroff, A. G., Thomas, J. A., & Anderson, A. A. Construction of an experimental sequential item test, Res. Memo. 60-1, The Adjutant General's Office, 1960.

7. Bean, K. L. Construction of educational and personnel tests. New York: McGraw-Hill, 1953.

8. Bloom, B. S. (Ed.) Taxonomy of educational objectives: the classification of educational goals. New York: Longmans, Green, 1956.

9. Brigham, C. C. A study of error. New York: College Entrance Examination Board, 1932.

10. Brogden, H. E., & Taylor, E. K. The Theory and classification of criterion bias. Educ. psychol. Measmt, 1950, 10, 159-186.

11. Brogden, H. E., & Taylor, E. K. The dollar criterion--applying the cost accounting concept to criterion construction. Personnel Psychol., 1950, 3, 133-154.

12. Brueckner, L. J., & Elwell, M. Reliability of diagnosis of error in multiplication of fractions. J. educ. Res., 1932, 26, 175-185.

13. Bruner, J. The act of discovery. Harv. educ. Rev., 1961, 31, 21-32.

14. Buswell, G. T., & John, L. Diagnostic studies in arithmetic, Suppl. Educ. Monogr., Univer. of Chicago, 1926, No. 30.

15. Carr, W. J. A functional analysis of self-instructional devices. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

16. Cline, V. B., Beals, A. R., & Seidman, D. Experimenting with accelerated training programs for men of various intelligence levels. Educ. psychol. Measmt, 1960, 20, 723-735.

17. Conrad, H. S. The experimental tryout of test materials. In Lindquist, E. F. (Ed.) Educational measurement. Washington, D. C.: Amer. Council on Education, 1951.

18. Cook, J. O. Response analysis in paired associate learning experiments. Raleigh, N. C.: North Carolina State College, 1960.

19. Cook, J. O., & Spitzer, M. E. Supplementary Report: Prompting vs. confirmation in paired-associate learning. J. exp. Psychol., 1960, 59, 275-276.

20. Coulson, J. E., & Silberman, H. F. Results of an initial experiment in automated teaching. In A. A. Lumsdaine and R. Glaser (Eds.) Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

21. Cronbach, L. J. Essentials of psychological testing. New York: Harper Bros., 1949.

22. Cronbach, L. J. Processes affecting scores on "Understanding of Others," and "Assumed Similarity." Psychol. Bull., 1955, 52, 177-193.

23. Cronbach, L. J., & Gleser, G. C. Psychological tests and personnel decisions. Urbana: Univer. of Illinois Press, 1957.

24. Cronbach, L. J., & Meehl, P. E. Construct validity in psychological tests. Psychol. Bull., 1955, 52, 281-302.

25. Crowder, N. A. The arithmetic of computers. Garden City, N. Y.: Doubleday, 1960.

26. Crowder, N. A. Intrinsic and extrinsic programming. Paper given at Conference on Application of Digital Computers to Automated Instruction, Washington, D. C., Oct., 1961.

27. Crowder, N. A. Automated tutoring by intrinsic programming. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

28. Dale, E., & Chall, J. S. A formula for predicting readability. Educ. res. Bull., 1948, 27, 11-20.

29. Davis, F. B., & Fifer, G. The effect on test reliability and validity of scoring aptitude and achievement tests with weights for every choice. Educ. psychol. Measmt, 1959, 19, 159-170.

30. DuBois, P. H. The design of correlational studies in training. In R. Glaser (Ed.) Training research and education. Pittsburgh, Pa.: Univer. Pittsburgh, 1961.

31. Dunn, T. F., & Goldstein, L. G. Test difficulty, validity, and reliability as functions of selected multiple item construction principles. Educ. psychol. Measmt, 1959, 19, 171-179.

32. Dyer, H. S., & King, R. G. College Board scores: their use and interpretation, No. 2. New York: College Entrance Examination Board, 1955.

33. Ebel, R. L. Writing the test item. In Lindquist, E. F. (Ed.) Educational measurement. Washington, D. C.: Amer. Council on Education, 1951.

34. Edwards, A. L. Techniques of attitude scale construction. New York: Appleton-Century-Crofts, 1957.

35. Eigen, L. D. Sets, relations, and functions: Book I. New York: Center for Programed Instruction, 1961.

36. Estes, W. K. Learning theory and the new "Mental Chemistry." Psychol. Rev., 1960, 67, 207-223.

37. Evans, J. L. An investigation of "Teaching Machine" variables using learning programs in symbolic logic. Doctoral dissertation, Univer. Pittsburgh, 1960.

38. Evans, J. L. The TMI-Grolier contributions. Paper presented at Amer. Psychol. Assoc. Convention, 1961.

39. Evans, J. L., Glaser, R., & Homme, L. E. A preliminary investigation of variation in the properties of verbal learning sequences of the "Teaching Machine" type. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

40. Evans, J. L., Glaser, R., & Homme, L. E. The RULEG System for the construction of programmed verbal learning sequences. Univer. Pittsburgh, 1960.

41. Ferris, F. Testing for physics achievement. Amer. J. Physics, 1960, 28, 269-278.

42. Ferster, C. B., & Sapon, S. M. An application of recent developments in psychology to the teaching of German. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

43. Flanagan, J. D. Units, scores, and norms. In Lindquist, E. F. (Ed.) Educational measurement. Washington, D. C.: Amer. Council on Education, 1951.

44. Flesch, R. A new readability yardstick. J. appl. Psychol., 1948, 32, 221-233.

45. Frederiksen, N. Development of the test "Formulating Hypotheses": a progress report. ONR Technical Report, Princeton, N. J.: Educational Testing Service, 1959.

46. Frederiksen, N., & Gilbert, A. C. F. Replication of a study of differential predictability. Educ. psychol. Measmt, 1960, 20, 759-767.

47. French, J. W. Validation of the SAT and new item types against four-year academic criteria. Research Bulletin 57-4. Princeton, N. J.: Educational Testing Service, 1957.

48. French, J. W. A machine search for moderator variables in massive data. ONR Technical Report, Princeton, N. J.: Educational Testing Service, 1961.

49. Fry, E. B. Teaching Machines: An investigation of constructed-response versus multiple-choice methods of response. Doctoral dissertation, Univer. Southern Calif., 1960.

50. Gagné, R. M. The effect of sequence of presentation of similar items on the learning of paired associates. J. exp. Psychol., 1950, 40, 61-73.

51. Gagné, R. M. Teaching machines and transfer of training. Paper presented at Amer. Psychol. Assoc. Convention, 1959.

52. Gagné, R. M., & Dick, W. Learning measures in a self-instructional program in solving equations. Princeton Univer., 1961.

53. Galanter, E. (Ed.) Automatic teaching: The state of the art. New York: Wiley, 1959.

54. Gavurin, E. I., & Donahue, V. M. Logical sequence and random sequence. Automated Teaching Bulletin, 1961, 1, 3-9.

55. Gilbert, T. F. On the relevance of laboratory investigation of learning to self-instructional programming. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

56. Gilbert, T. F. Some recent attempts at the partial automation of teaching. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

57. Glaser, R. Principles and problems in the preparation of programmed learning sequences. Univer. Pittsburgh, 1960.

58. Goldbeck, R. A. The effect of response mode and learning material difficulty on automated instruction. Santa Barbara, Calif.: Amer. Institute for Research, 1960.

59. Goldbeck, R. A., Campbell, V. N., & Llewellyn, J. E. Further experimental evidence on response modes in automated instruction. Santa Barbara, Calif.: Amer. Institute for Research, 1960.

60. Gorow, F. F. Do it yourself statistics. Long Beach, Calif.: Long Beach State College, 1960.

61. Green, B. F. Attitude measurement. In Lindzey, G., Handbook of Social Psychology, Cambridge, Mass.: Addison-Wesley, 1954.

62. Grossnickle, F. E. Constancy of error in learning division with a two-figure divisor. J. educ. Res., 1939, 33, 189-196.

63. Grossnickle, F. E. Kinds of errors in division of decimals and their constancy. J. educ. Res., 1944, 37, 110-117.

64. Guilford, J. P. Psychometric methods. New York: McGraw-Hill, 1954.

65. Gulliksen, H. Theory of mental tests. New York: Wiley, 1950.

66. Guttman, L. A basis for scaling quantitative data. Amer. sociol. Rev.,1949, 9, 139-150.

67. Heim, A. W. Adaptation to level of difficulty in intelligence testing. Brit. J. Psychol., 1955, 46, 211-224.

68. Holland, J. G. A teaching machine program in psychology. In Galanter, E. (Ed.) Automatic teaching: The state of the art. New York: Wiley, 1959.

69. Holland, J. G. Teaching machines: an application of principles from the laboratory. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

70. Holland, J. G. Design and use of a teaching machine program. Paper presented at Amer. Psychol. Assoc. Convention, 1960.

71. Holland, J. G., & Skinner, B. F. The analysis of behavior. New York: McGraw-Hill, 1961.

72. Hosmer, C. L. Payoff in programming of tutoring units. Paper presented at Amer. Psychol. Assoc. Convention, 1961.

73. Hughes, J. L. The effectiveness of programmed instruction: experimental findings for 7070 training. IBM, 1961.

74. Hutt, M. L. A clinical study of "Consecutive" and "Adaptive" testing with the revised Stanford-Binet. J. consult. Psychol., 1947, 11, 93-103.

75. Irion, A. L., & Briggs, L. J. Learning task and mode of operation variables in use of the subject matter trainer. Tech. Report AFPTRC-TR-57-8, 1957.

76. Jackson, D. N., & Messick, S. Content and style in personality assessment. Psychol. Bull., 1958, 55, 243-252.

77. Jacobs, P. I. Item difficulty and programmed learning. Research Memorandum 61-5. Princeton, N. J.: Educational Testing Service, 1961.

78. Jenkins, J. J. In R. O. Collier, Jr., and S. M. Elam (Eds.), Research design and analysis. Bloomington, Ind.: Phi Delta Kappa, Inc., 1961.

79. Jones, M. B. Simplex theory, Monograph 3, U. S. Naval School of Aviation Medicine, 1959.

80. Jones, R. S. Integration of instructional with self-scoring measuring procedures. Unpublished doctoral dissertation, Ohio State Univer., 1950.

81. Kaess, W., & Zeaman, D. Positive and negative knowledge of results on a Pressey-type punchboard. J. exp. Psychol., 1960, 60, 12-17.

82. Keislar, E. R. The development of understanding in arithmetic by a teaching machine. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

83. Kendler, H. H. Teaching machines and psychological theory. In Galanter, E. (Ed.) Automatic teaching: The state of the art. New York: Wiley, 1959.

84. Kershaw, J. A., & McKean, R. N. Systems analysis and education. RM-2473-FF, The Rand Corporation, 1959.

85. Kimble, G. A., & Dufort, R. H. Meaningfulness and isolation as factors in verbal learning. J. exp. Psychol., 1955, 50, 361-368.

86. Klaus, D. J. The art of auto-instructional programming. Audio-Visual Comm. Rev., 1961, 9, 130-142.

87. Kopstein, F. F., & Roshal, S. M. Verbal learning efficiency as a function of the manipulation of representational response processes. To appear in A. A. Lumsdaine (Ed.) Student response in programmed instruction. Washington, D. C.: National Academy of Sciences, National Research Council (in press).

88. Kopstein, F. F., & Shillestad, I. J. A survey of auto-instructional devices. USAF, Aeronautical Systems Division, Wright-Patterson AFB, Ohio, ASD Tech. Rep. 61-414, 1961.

89. Krathwohl, D. R., & Huyser, R. J. The sequential item test. Paper presented at Amer. Psychol. Assoc. Convention, 1956.

90. Lawson, C. A., Burmester, M. A., & Nelson, C. H. Developing a scrambled book and measuring its effectiveness as an aid to learning natural science. Science Educ., 1960, 44, 347-358.

91. Lindquist, E. F. (Ed.) Educational measurement. Washington, D. C.: Amer. Council on Education, 1951.

92. Lindquist, E. F. Preliminary considerations in objective test construction. In Lindquist, E. F. (Ed.) Educational measurement. Washington, D. C.: Amer. Council on Education, 1951.

93. Lord, F. M. Estimating test reliability. Educ. psychol. Measmt, 1955, 15, 325-336.

94. Lumsdaine, A. A. Some issues concerning devices and programs for automated learning. In. A. A. Lumsdaine and R. Glaser (Eds.), Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

95. Lumsdaine, A. A., & Glaser, R. Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

96. Lund, K. W. Test performance as related to the order of item difficulty, anxiety, and intelligence. Unpublished doctoral thesis, Northwestern Univer., 1953.

97. Mager, R. F. Preparing objectives for programmed instruction. San Francisco: Fearon Publishers, 1961.

98. Messick, S., & Hills, J. R. Objective measurement of personality: cautiousness and intolerance of ambiguity. Educ. psychol. Measmt, 1960, 20, 685-698.

99. Meyer, S. R. Report on the initial test of a Junior High School vocabulary program. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

100. Mitzel, H. E. Teacher effectiveness. In Harris, C. W. (Ed.) Encyclopedia of Educational Research. New York: Macmillan, 1960.

101. Mosier, C. I. A critical examination of the concepts of face validity. Educ. psychol. Measmt, 1947, 7, 191-205.

102. Nachman, M., & Opochinsky, S. The effects of different teaching methods. J. educ. Psychol., 1958, 49, 245-249.

103. Newman, S. E. Student vs. instructor design of study method. J. educ. Psychol., 1957, 48, 328-333.

104. Pask, G. Electronic keyboard teaching machines. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

105. Porter, D. Some effects of year long teaching-machine instruction. In Galanter, E. (Ed.) Automatic teaching: The state of the art. New York: Wiley, 1959.

106. Pressey, S. L. Development and appraisal of devices providing immediate automatic scoring of objective tests and concomitant self-instruction. J. Psychol., 1950, 29, 417-447.

107. Pressey, S. L., & Campbell, P. The causes of children's errors in capitalization. The English Journal, 1933, 22, 197-201.

108. Rao, R. C. Review of simplex theory. Psychometrika, 1961, 26, 252-254.

109. Rapaport, D. Diagnostic psychological testing, Vol. 1. Chicago: The Year Book Publishers, Inc., 1945.

110. Remmers, H. H., & Gage, N. L. Educational measurement and evaluation. New York: Harper, 1943.

111. Rigney, J. W., & Fry, E. B. Current teaching-machine programs and programming techniques. A-V Comm. Rev. 1961, Supplement 3.

112. Riley, M. W., Riley, J. W., Jr., & Toby, J. Sociological studies in scale analysis. New Brunswick: Rutgers Univer. Press, 1954.

113. Roe, A. Automated teaching methods using linear programs, Report No. 60-105, UCLA, 1960.

114. Rothkopf, E. Z. Automated teaching devices and a comparison of two variations of the method of adjusted learning. Psychol. Rep., 1961, 8, 163-169.

115. Rothkopf, E. Z. Criteria for the acceptance of self-instructional programs. Paper read at Educational Records Bureau Meeting, New York, 1961.

116. Ryans, D. G., & Frederiksen, N. Performance tests of educational achievement. In Lindquist, E. F. (Ed.) Educational measurement. Washington, D. C.: Amer. Council on Education, 1951.

117. Saunders, D. R. Moderator variables in predication. Educ. psychol. Measmt, 1956, 16, 209-222.

118. Schultz, D. G., & Siegel, A. I. Generalized Thurstone and Guttman scales for measuring technical skills in job performance. J. appl. Psychol., 1961, 45, 137-142.

119. Severin, D. G. Appraisal of special tests and procedures used with self-scoring instruction testing devices. Unpublished doctoral thesis, Ohio State Univer., 1951.

120. Shay, C. B. The relationship of intelligence to size of step on a teaching machine program. Doctoral dissertation, UCLA, 1960.

121. Silberman, H. F., Melaragno, R. J., Coulson, J. E., & Estavan, D. Fixed sequence vs. branching auto-instructional methods. J. educ. Psychol., 1961, 52, 166-172.

122. Silverman, R. E., & Alter, M. Response mode, pacing, and motivational effects in teaching machines. Tech. Rep. NAVTRADEVCEN 507-3, 1961.

123. Skinner, B. F. Teaching machines. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

124. Skinner, B. F. Why we need teaching machines. Harv. Educ. Rev., 1961, 31, 379-398.

125. Skinner, B. F., & Holland, J. G. The use of teaching machines in college instruction. In Lumsdaine and Glaser, Teaching machines and programmed learning: a source book. Washington, D. C.: National Educational Association, 1960.

126. Stephens, A. L. Certain special factors involved in the law of effect. Unpublished doctoral thesis, Ohio State Univer., 1950.

127. Stolurow, L. M. Teaching by machine. Washington, D. C.: U. S. Office of Education, 1961.

128. Swineford, F., & Miller, P. M. Effects of directions regarding guessing on item statistics of a multiple-choice vocabulary test. J. educ. Psychol., 1953, 44, 129-139.

129. Taber, J. I., Glaser, R., & Schaefer, H. H. Instructional programming--a handbook of instructional procedures. Univer. Pittsburgh, draft, 1961.

130. Tate, M. W. Individual differences in speed of response in mental test materials of varying degrees of difficulty. Educ. psychol. Measmt, 1948, 8, 353-374.

131. Thorndike, R. L. Personnel selection: test and measurement techniques. New York: Wiley, 1949.

132. Thorndike, R. L. Reliability. In Lindquist, E. F. (Ed.) Educational measurement. Washington, D. C.: Amer. Council on Education, 1951.

133. Torgerson, W. S. Theory and methods of scaling. New York: Wiley, 1958.

134. Travers, R. M. W. How to make achievement tests. New York: Odyssey Press, 1950.

135. Traxler, A. Administering and scoring the objective test. In Lindquist, E. F. (Ed.) Educational measurement. Washington, D. C.: Amer. Council on Education, 1951.

136. Vaughn, K. W. Planning the objective test. In Lindquist, E. F. (Ed.) Educational measurement. Washington, D. C.: Amer. Council on Education, 1951.

INDEX OF NAMES

SUBJECT INDEX